

# Systèmes d'exploitation : principes, programmation et virtualisation

---

## Table des matières

<b>1</b>	<b>OBJECTIFS</b>	<b>3</b>
1.1	OBJECTIFS VISES	3
1.2	PLAN	3
<b>2</b>	<b>RAPPELS SUR LA STRUCTURE D'UN ORDINATEUR</b>	<b>4</b>
2.1	ARCHITECTURE GENERALE D'UN SYSTEME INFORMATIQUE	4
2.2	LE PROCESSEUR	4
2.3	LE SYSTEME MEMOIRE	6
2.4	LE SYSTEME D'ENTREES SORTIES	6
2.5	EXECUTION MULTIMODE	8
<b>3</b>	<b>PRINCIPES GENERAUX D'UN SYSTEME D'EXPLOITATION</b>	<b>9</b>
3.1	FONCTIONS D'UN SYSTEME D'EXPLOITATION	9
3.2	CLASSES DE SYSTEMES	10
3.3	UNIX ET LINUX	11
<b>4</b>	<b>RAPPEL DE PROGRAMMATION EN C</b>	<b>12</b>
4.1	CHAINE DE PRODUCTION DES EXECUTABLES	12
4.2	ILLUSTRATION	14

# Systèmes d'exploitation : principes, programmation et virtualisation

---

## 1 Objectifs

### 1.1 Objectifs visés

Permettre de comprendre les principaux concepts et paradigmes et les mécanismes mis en œuvre dans les systèmes d'exploitation modernes, les mécanismes mis en œuvre dans le noyau des systèmes tels que Linux ainsi que les nouvelles architectures matérielles gérées par ces systèmes (processeurs multicœurs, support de la virtualisation de systèmes).

Obtenir des bases dans la programmation concurrente et dans la compréhension des mécanismes de gestion du parallélisme utilisés dans les noyaux des systèmes.

### 1.2 Plan

Le programme de l'ensemble du cours est inspiré de celui de M. Ivan KURZWEG et est le suivant :

- Présentation des systèmes d'exploitation, rappels sur la programmation C en environnement Unix
- Processus et Threads, gestion et ordonnancement
- Gestion de la mémoire
- Pagination de la mémoire virtuelle
- Gestion de fichiers
- Exclusion mutuelle
- Interblocage
- Moniteurs

## 2 Rappels sur la structure d'un ordinateur

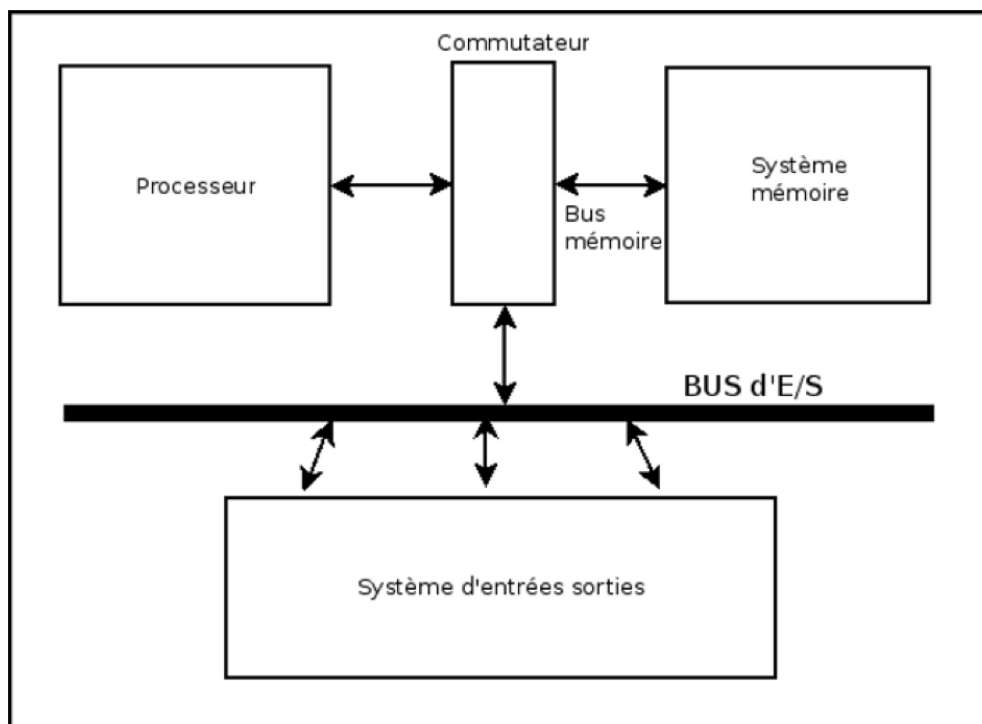
Ce chapitre rappelle les bases de l'architecture matérielle des ordinateurs, exposant ainsi les problématiques auxquelles les systèmes d'exploitation modernes doivent faire face.

### 2.1 Architecture générale d'un système informatique

La plupart des systèmes informatiques peuvent être divisés en trois sous-systèmes :

- Processeur ;
- Mémoire ;
- Sous-système d'entrées-sorties.

La figure ci-dessous présente les relations entre ces sous-systèmes :

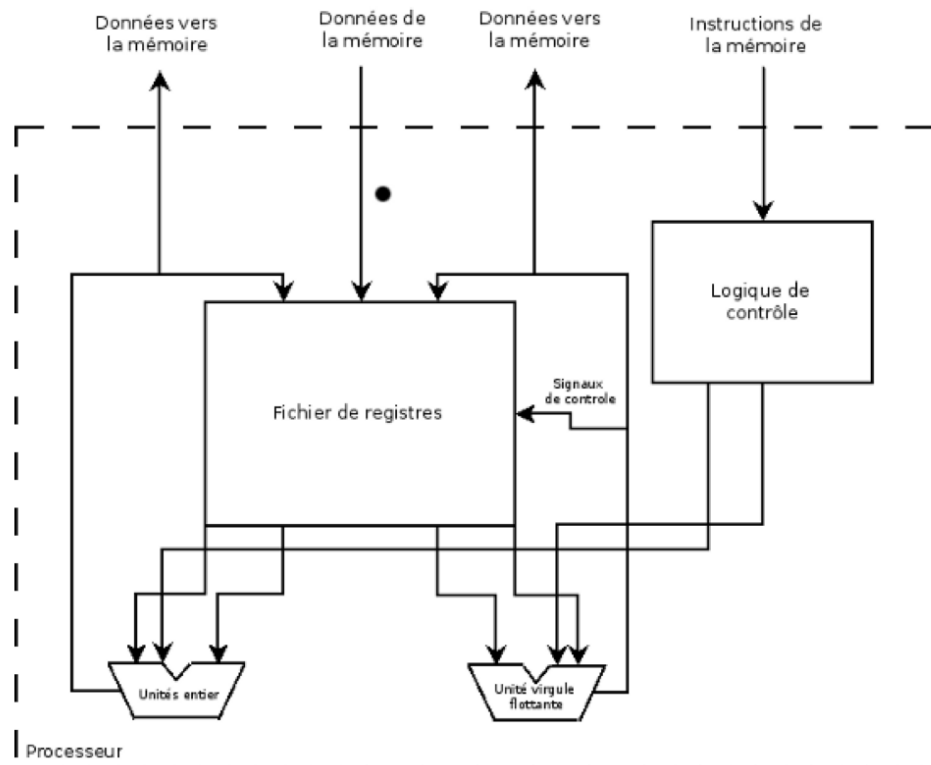


*Fig. 1 : structure d'un ordinateur*

Le processeur a pour charge d'exécuter les programmes, la mémoire fournit l'espace de stockage, tandis que le sous-système d'E/S permet le contrôle des appareils tels que disques durs, lecteurs optiques, cartes vidéos, ... Le commutateur (il peut être intégré au processeur) permet les échanges de données via les bus d'E/S et mémoire. Cette structure est globalement la même depuis sa définition par John Von Neuman en 1945.

### 2.2 Le processeur

Les processeurs sont constitués de plusieurs blocs : unités d'exécutions, fichiers de registres et logique de contrôle. Les unités d'exécution contiennent le dispositif matériel qui exécute les instructions : les *Unités Arithmétiques et Logiques* (UAL), capables d'effectuer les opérations élémentaires habituelles sur des valeurs binaires (addition, soustraction, ET, OU, ...)



**Fig. 2 : diagramme de bloc d'un processeur**

Le *fichier de registres* est une zone de stockage que le processeur utilise pour stocker des données, de manière plus efficace que le stockage en mémoire centrale. Les registres sont ainsi généralement utilisés pour mémoriser des résultats intermédiaires ou des états particuliers du processeur. La *logique de contrôle* contrôle le reste du processeur, en déterminant le moment auquel les instructions sont exécutées et quelles opérations doivent être exécutées pour chaque instruction.

Les instructions sont en général assez rudimentaires : ce sont essentiellement des opérations de transfert de données entre les registres et l'extérieur du processeur (mémoire ou périphérique), ou des opérations arithmétiques ou logiques avec un ou deux opérandes. Pour ces dernières opérations un registre particulier, l'accumulateur, est souvent utilisé implicitement comme l'un des opérandes et comme résultat. En général le déroulement de l'instruction entraîne l'incrémentement du compteur ordinal, et donc l'exécution de l'instruction qui suit.

La tendance naturelle a été de construire des processeurs avec un jeu d'instructions de plus en plus large; on pensait alors que le programmeur utiliserait les instructions ainsi disponibles pour améliorer l'efficacité de ses programmes. Ceci a conduit à ce que l'on a appelé l'architecture CISC (*Complex Instruction Set Computer*). Cependant on a constaté que les programmes contenaient toujours les mêmes instructions, une partie importante du jeu d'instructions n'étant utilisée que très rarement. Une nouvelle famille de processeurs a alors été construite, l'architecture RISC (*Reduced Instruction Set Computer*), qui offre un jeu réduit d'instructions simples mais très rapides.

## 2.3 Le système mémoire

Le système mémoire agit à la manière d'un réceptacle de stockage pour les données et les programmes. La mémoire centrale (mémoire *vive*, ou encore *mémoire à accès aléatoire*, RAM) est divisée en une série d'emplacements de stockage, chacun étant numéroté (son *adresse*). L'une des caractéristiques des systèmes informatiques est ainsi la largeur des adresses qu'ils utilisent, car elle limite la quantité de mémoire utilisable (on parle de systèmes 32 bits, ou 64 bits).

La conception du système mémoire possède un impact déterminant sur la performance du système informatique, et se trouve souvent être le facteur limitant pour la vitesse d'exécution des applications. La bande passante et la latence sont capitales pour la performance des applications. Nous verrons également dans le chapitre qui lui est consacrée que la gestion de la mémoire implique également des notions de protection et de mécanisme de partage entre les processus.

## 2.4 Le système d'entrées sorties

Les entrées sorties permettent l'échange de données entre deux constituants physiques, et nécessitent trois liaisons :

- les données elles mêmes ;
- une liaison permettant à l'émetteur de signaler la présence de la donnée ;
- une liaison permettant au récepteur de signaler qu'il a lu la donnée.

Il existe trois manières d'implémenter ce principe : le mode programmé, le mode DMA, et les processeurs spécialisés.

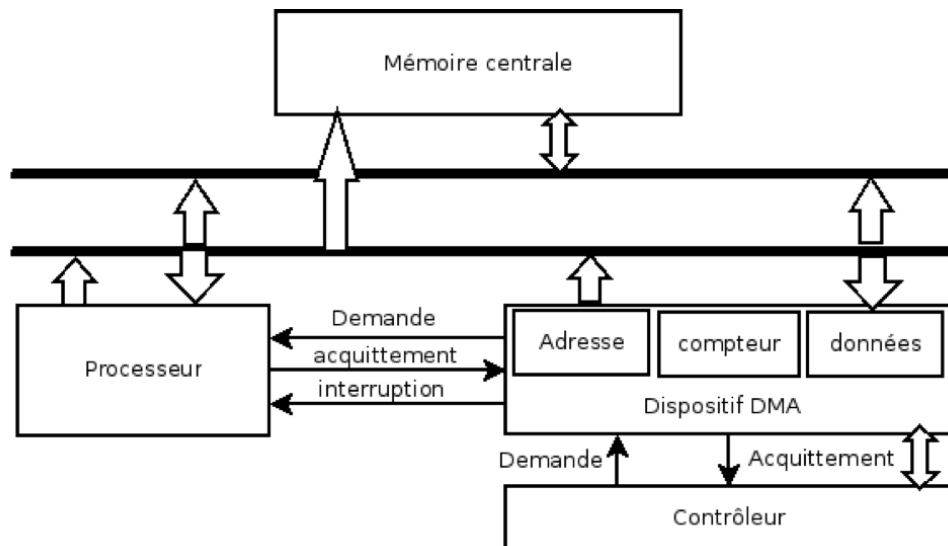
### 2.4.1 Le mode programmé

Dans ce mode, l'unité d'échange qui interface le périphérique au bus et au processeur, ne sait pas délivrer d'informations sur son état. Pour savoir si l'unité d'échange est prête à recevoir ou délivrer une nouvelle donnée, le processeur doit régulièrement lire le contenu du registre d'état de l'unité d'échange. Le processeur est donc complètement occupé durant l'entrée sortie par la réalisation de cette boucle de scrutation dont la logique est donnée ci-dessous suivante en mémoire centrale. On ne peut donc concevoir qu'un système monoprogrammé dans ce cas.

```
tantque il_y_a_des_données_à_lire faire
    tantque donnée_suivante_non_prête faire fait; { attente de la donnée}
    lire_la_donnée;
    traitement_de_la_donnée;
fait
```

### 2.4.2 Accès direct à la mémoire

Pour accroître le débit des entrées-sorties, et diminuer la monopolisation du processeur dont nous avons parlé ci-dessus, la première solution a été de déporter un peu de fonctionnalités dans le dispositif qui relie le périphérique au bus, de façon à lui permettre de ranger directement les données provenant du périphérique en mémoire dans le cas d'une lecture, ou d'extraire directement ces données de la mémoire dans le cas d'une écriture. C'est ce que l'on appelle l'accès *direct à la mémoire*, DMA (Direct Memory Access)



**Fig. 3 : dispositif DMA**

L'exécution d'un transfert se déroule de la manière suivante :

- 1) Le processeur informe le dispositif d'accès direct à la mémoire de l'adresse en mémoire où commence le tampon qui recevra les données s'il s'agit d'une lecture, ou qui contient les données s'il s'agit d'une écriture. Il informe également le dispositif du nombre d'octets à transférer.
- 2) Le processeur informe le contrôleur des paramètres de l'opération proprement dite.
- 3) Pour chaque donnée élémentaire échangée avec le contrôleur, le dispositif demande au processeur le contrôle du bus, effectue la lecture ou l'écriture mémoire à l'adresse contenue dans son registre et libère le bus. Il incrémente ensuite cette adresse et décrémente le compteur.
- 4) Lorsque le compteur atteint zéro, le dispositif informe le processeur de la fin du transfert par une ligne d'interruption.

Pendant toute la durée du transfert, le processeur est libre d'effectuer une autre tâche.

### 2.4.3 Processeur spécialisé

La troisième façon de relier un périphérique avec la mémoire est d'utiliser un processeur spécialisé d'entrées-sorties : déléguer plus d'automatisme à ce niveau. Dans le schéma précédent, le processeur principal avait en charge la préparation de tous les dispositifs, accès direct à la mémoire, contrôleur, périphérique, etc..., pour la réalisation de l'opération. L'utilisation d'un processeur spécialisé a pour but de reporter dans ce processeur la prise en charge de cette préparation, mais aussi des opérations plus complexes, telles que par exemple le contrôle et la reprise d'erreurs. L'intérêt est de faire exécuter des tâches de bas niveau par un processeur moins performant, et donc moins coûteux à réaliser, tout en réservant le processeur principal à des tâches plus "nobles".

### 2.4.4 Les interruptions

Une interruption est un signal envoyé au processeur par un périphérique extérieur. Sorte de "bi-peur" du processeur, elle indique à ce dernier d'interrompre l'ensemble de ses activités en cours

pour répondre aux besoins du périphérique à l'origine de l'interruption. Le processeur termine le traitement de l'instruction en cours, et contrôle les interruptions.

Le processeur répond à l'interruption en stockant la valeur en cours du compteur ordinal, qui lui permettra de reprendre l'exécution au point où l'interruption s'est produite. La valeur du compteur ordinal (et d'autres indicateurs complémentaires) est stockée dans un registre d'état, le PSW (*Program Status Word*). Le contenu de l'ensemble des registres est également sauvegardé, c'est ce que l'on appelle la sauvegarde du contexte.

Enfin, les interruptions peuvent être associées à un niveau de priorité matérielle, et masquées. Ceci permet de résoudre des cas d'interruptions survenant lors du traitement d'une précédente interruption. Effectivement, sans ces mécanismes, le contexte d'un programme pourrait être perdu.

## 2.5 Exécution multimode

Nous venons de voir qu'il est nécessaire de contrôler les actions d'un programme vis-à-vis de son environnement direct. On doit donc restreindre les privilèges des programmes, et confier au système d'exploitation des activités réservées (exécution d'instructions privilégiées, accès à certains registres, à certains périphériques d'E/S, ...). Les processeurs possèdent donc deux modes de fonctionnement, le *mode superviseur* (ou *maître*), et le *mode utilisateur* (ou *esclave*). Il existe trois manières de passer en mode superviseur :

- une instruction spéciale : l'appel système ou Supervisor Call (SVC)
- les interruptions matérielles (IRQ) : asynchrone
- les déroutements (conditions anormales détectés par le processeur) : interruption de type exception (synchrone)

Dans le mode utilisateur, chaque instruction est contrôlée avant son exécution, pour vérifier si elle est autorisée.



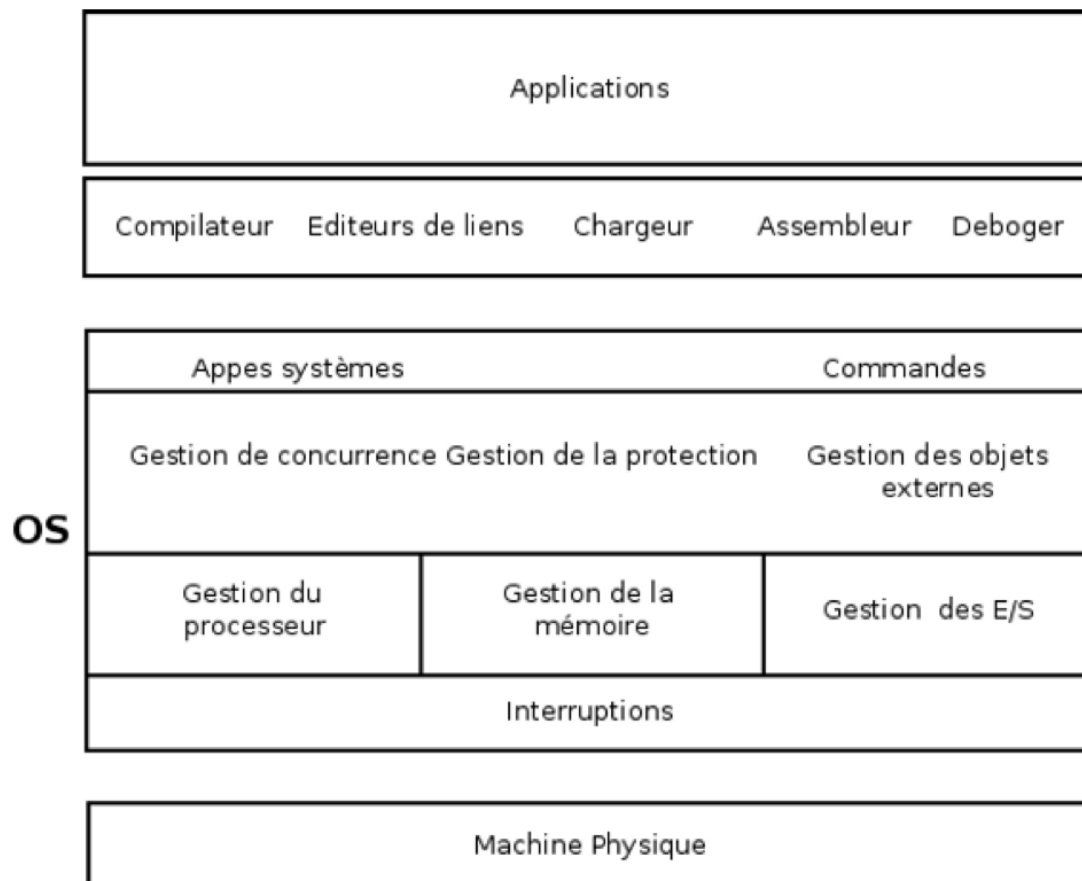
### 3 Principes généraux d'un système d'exploitation

Le système d'exploitation est le logiciel qui prend en charge les fonctionnalités élémentaires du matériel et qui propose une plate-forme plus efficace en vue de l'exécution des programmes. Il gère les ressources matérielles, offre des services pour accéder à ces ressources, et crée des éléments abstraits de niveau supérieur, tels que des fichiers, des répertoires et des processus.

#### 3.1 Fonctions d'un système d'exploitation

Le système d'exploitation se présente donc comme une couche logicielle placée entre la machine matérielle et les applications. Il s'interface avec la couche matérielle, notamment par le biais du mécanisme des interruptions. Il s'interface avec les applications par le biais des primitives qu'il offre : appels système et commandes. Le système d'exploitation peut être découpé en plusieurs grandes fonctions :

- **Gestion du processeur** : le système doit gérer l'allocation du processeur aux différents programmes pouvant s'exécuter. Cette allocation se fait par le biais d'un algorithme d'ordonnancement qui planifie l'exécution des programmes. Selon le type de système d'exploitation, l'algorithme d'ordonnancement répond à des objectifs différents
- **Gestion de la mémoire** : le système doit gérer l'allocation de la mémoire centrale entre les différents programmes pouvant s'exécuter. Comme la mémoire physique est parfois trop petite pour contenir la totalité des programmes, la gestion de la mémoire se fait selon le principe de la mémoire virtuelle : à un instant donné, seules sont chargées en mémoire centrale, les parties de code et données utiles à l'exécution
- **Gestion des entrées/sorties** : le système doit gérer l'accès aux périphériques, c'est-à-dire faire la liaison entre les appels de haut niveau des programmes utilisateurs et les opérations de bas niveau de l'unité d'échange responsable du périphérique (unité d'échange clavier) : c'est le pilote d'entrées/sorties (driver) qui assure cette correspondance
- **Gestion de la concurrence** : Comme plusieurs programmes coexistent en mémoire centrale, ceux-ci peuvent vouloir communiquer pour échanger des données. Par ailleurs, il faut synchroniser l'accès aux données partagées afin de maintenir leur cohérence. Le système offre des outils de communication et de synchronisation entre programmes
- **Gestion des objets externes** : La mémoire centrale est une mémoire volatile. Aussi, toutes les données devant être conservées au-delà de l'arrêt de la machine, doivent être stockées sur une mémoire de masse (disque dur, disquette, cédérom...). La gestion de l'allocation des mémoires de masse ainsi que l'accès aux données stockées s'appuient sur la notion de fichiers et de système de gestion de fichiers (SGF).
- **Gestion de la protection** : le système doit fournir des mécanismes garantissant que ses ressources (CPU, mémoire, fichiers) ne peuvent être utilisées que par les programmes auxquels les droits nécessaires ont été accordés. Il faut notamment protéger le système et la machine des programmes utilisateurs (mode d'exécution utilisateur et superviseur)



*Fig. 4 : fonctions d'un OS*

### 3.2 Classes de systèmes

– Les premiers temps de l'informatique ne permettaient guère aux utilisateurs d'avoir un accès direct aux ordinateurs. Les entrées (programmes et données) étaient préparées sur bande ou carte perforée. Les utilisateurs donnaient leur entrée à un opérateur, et récupéraient la sortie plus tard. L'opérateur assemblait les tâches similaires par lots, puis les exécutaient par le biais de l'ordinateur. Chaque tâche disposait d'un contrôle total de la machine pendant toute son exécution. Le système d'exploitation de ce type de machine est en mode de *traitement par lots* (*batch mode*). Les fonctionnalités de ces systèmes sont souvent minimales, sauf quand ils offrent en plus un service de temps partagé :

– Les systèmes par *lots en multiprogrammation* (*multiprogram batch system*) lisent les tâches à exécuter sur le disque. Lorsqu'une tâche n'est pas en mesure de s'exécuter (attente d'une E/S), un autre tâche prend le relais.

– Les systèmes à temps partagé (*time-shared*), ou encore systèmes multiutilisateurs interactifs, autorisent des interactions entre l'utilisateur et le processus. Dans les systèmes par lots, toutes les données sont fournies au moment où le programme est saisi. Lorsqu'une interaction avec l'utilisateur est nécessaire, le système d'exploitation doit gérer un environnement qui permet aux programmes de répondre aux saisies dans un laps de temps

convenable. Le système d'exploitation doit non seulement partager des ressources, mais il doit agir comme si les processus s'exécutaient simultanément. Pour cela il bascule très rapidement d'un processus actif à un autre.

– La mise en réseau des ordinateurs implique aujourd'hui des communications via des protocoles de plus en plus complexes. On parle donc de système d'exploitation *en réseau* (networked). Le système d'exploitation universel d'aujourd'hui est donc un système à temps partagé en réseau.

– Un système d'exploitation en *temps réel* (real-time) est conçu pour prendre en charge l'exécution de tâches dans le cadre de contraintes liées à l'horloge. Le système d'exploitation doit ainsi garantir que la tâche peut-être exécutée dans un délai spécifié. Comme ces systèmes sont souvent interfacés à un environnement dynamique (procédé) délivrant des événements synchrones ou asynchrones auxquels ils doivent réagir, on parle aussi de systèmes réactifs. Ce sont des systèmes adaptés à la commande de procédé.

– Les systèmes d'exploitation *répartis* (distributed) gèrent l'ensemble des ressources des machines en réseau. Les système d'exploitation de toutes les machines travaillent donc en collaboration pour gérer les ressources collectives. Un seul système d'exploitation gère les ressources du réseau, qui sont fournies par chaque ordinateur, ou *nœud*.

### 3.3 Unix et Linux

#### 3.3.1 Philosophie

##### *Tout est fichier*

Dans le monde Unix, les périphériques sont des fichiers qui se trouvent dans /dev, les processus sont des fichiers qui se trouvent dans /proc, les paramètres et données du noyau sont des fichiers qui se trouvent dans /sys et accessoirement /proc, les exécutables sont des fichiers normaux. Une telle vision des choses apporte énormément de souplesse dans l'approche de la programmation.

##### *Les données sont du texte*

Dans la philosophie unix, toutes les données doivent être stockées et transmises sous forme de texte, ce qui apporte de nombreux avantages :

- Un fichier texte peut être lu par les autres outils Unix, et donc respecte de principe faire une chose et le faire bien (cf. ci-dessous)
- Un fichier texte peut être lu par un être humain et donc respecte le principe KISS (cf. ci-dessous)
- Un fichier texte permet une interopérabilité avec d'autres systèmes
- Un fichier texte facilite le débogage

##### *KISS - Keep it simple, stupid !*

Le principe est de disposer d'outils très simples, faciles à comprendre, à manipuler, et à maintenir. Le code source doit également rester simple, "stupide" ... Un tel code, même sans trop de commentaires, sera simple à lire, à maintenir et à corriger.

### Faire une chose et le faire bien

Unix a inventé le pipe, une méthode simple pour faire communiquer deux processus entre eux. Cette communication est unidirectionnelle et non formatée, seules des données brutes y passent.

S'y ajoute les sockets, qui permettent une communication bidirectionnelle, et d'autres moyens de communication interprocessus. Ces mécanismes permettent de développer des outils sachant facilement communiquer avec ceux existants, qui normalement savent faire UNE chose, mais bien...

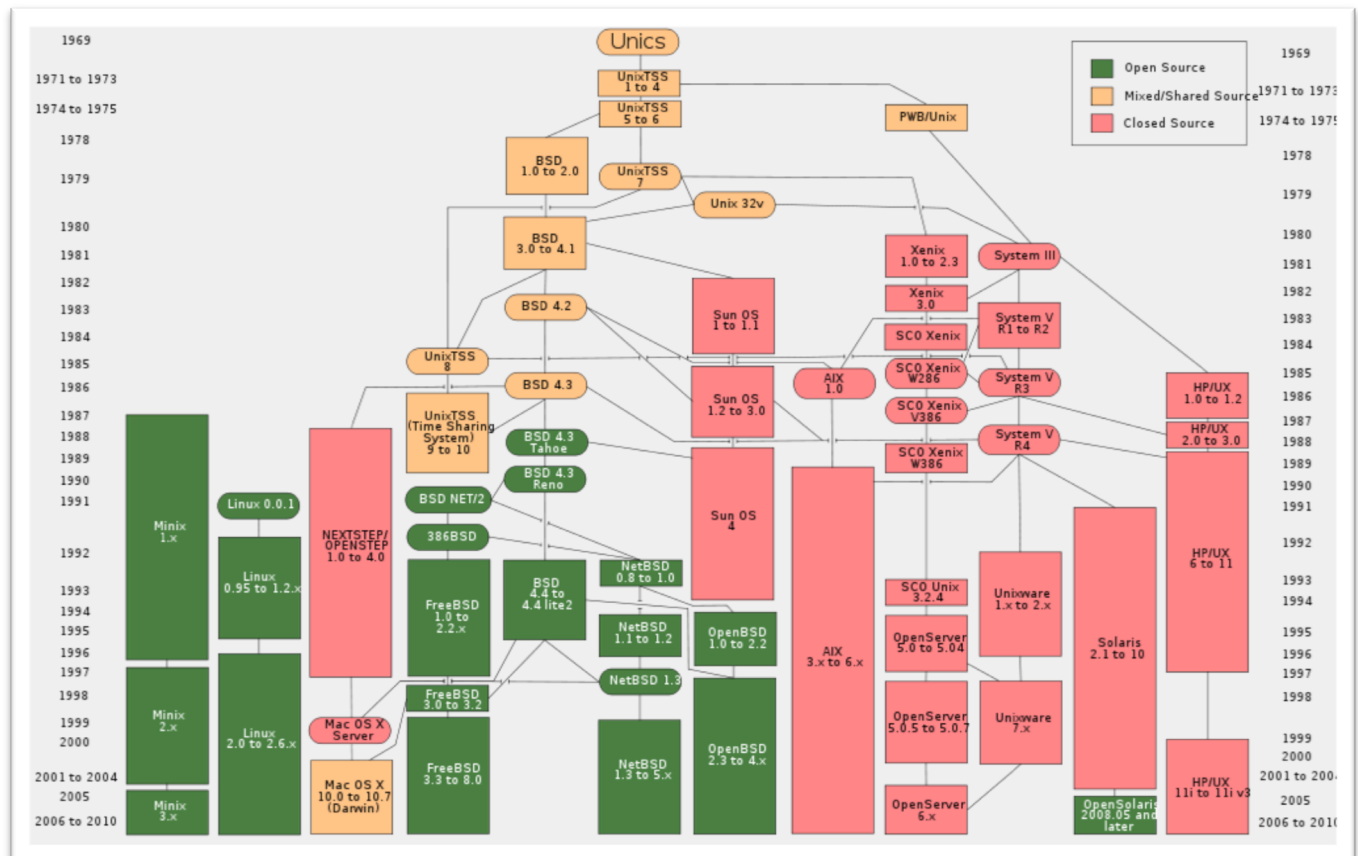


Fig. 5 : histoire des Unix (Wikipedia)

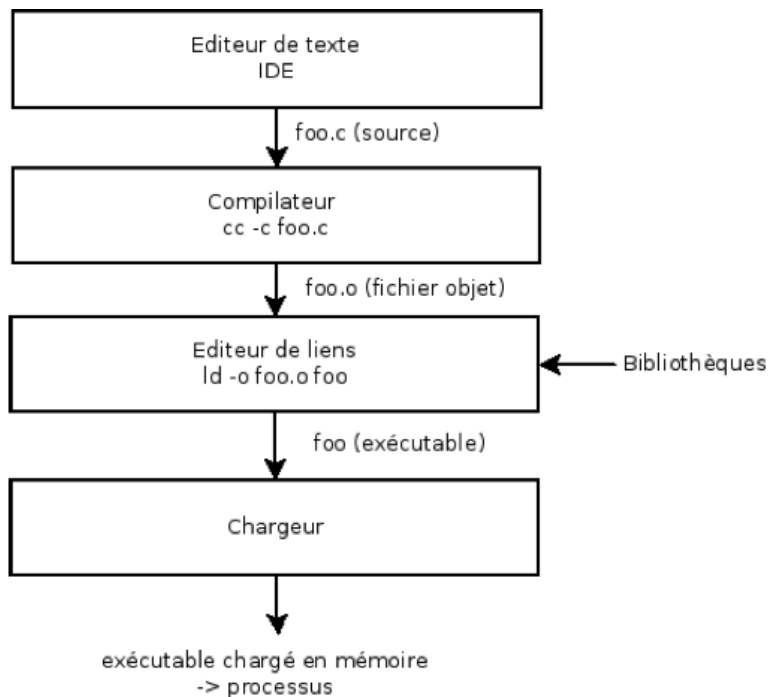
## 4 Rappel de programmation en C

### 4.1 Chaîne de production des exécutables

La production de programmes exécutables sur un système d'exploitation passe par plusieurs étapes, et permet de transformer du code écrit dans un langage de haut niveau vers un programme écrit en code machine :

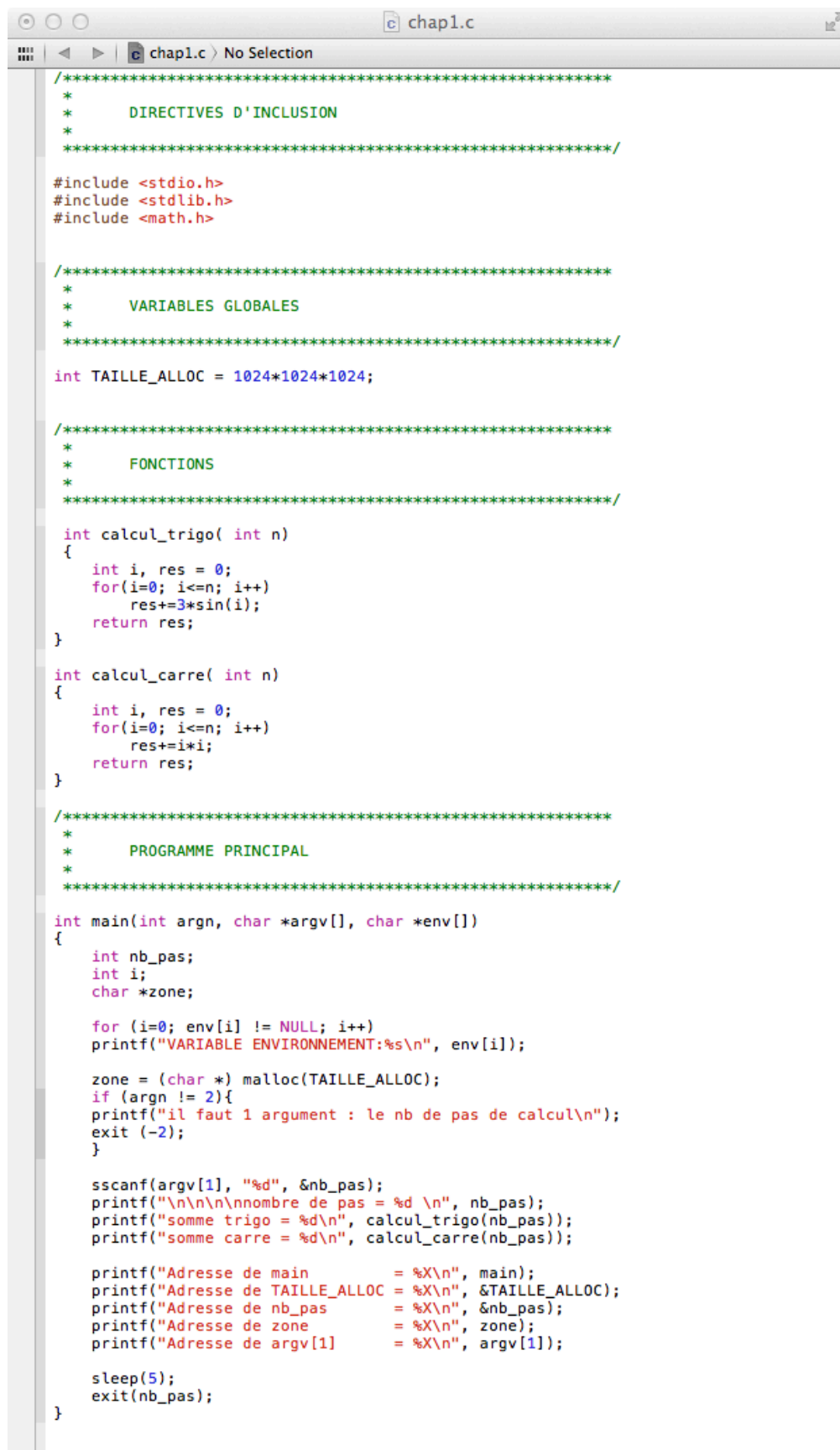
1. Saisie du programme dans le langage et enregistrement sur le disque : c'est le *fichier source*
2. Compilation avec le compilateur du langage (vérification syntaxique et lexicale) : on obtient le *fichier objet*
3. Edition de liens (résolution des références externes) : c'est le *fichier exécutable*

Lorsque l'utilisateur demande l'exécution de son programme, le fichier exécutable est alors monté en mémoire centrale : c'est l'étape de chargement. Le système alloue de la place mémoire pour placer le code et les données du programme. Nous précisons dans la partie suivante du cours cette dernière étape.



**Fig. 6 : chaîne de production d'un exécutable**

## 4.2 Illustration



```

/*****
 *
 *   DIRECTIVES D'INCLUSION
 *
 *****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*****
 *
 *   VARIABLES GLOBALES
 *
 *****/

int TAILLE_ALLOC = 1024*1024*1024;

/*****
 *
 *   FONCTIONS
 *
 *****/

int calcul_trigo( int n)
{
    int i, res = 0;
    for(i=0; i<=n; i++)
        res+=3*sin(i);
    return res;
}

int calcul_carre( int n)
{
    int i, res = 0;
    for(i=0; i<=n; i++)
        res+=i*i;
    return res;
}

/*****
 *
 *   PROGRAMME PRINCIPAL
 *
 *****/

int main(int argn, char *argv[], char *env[])
{
    int nb_pas;
    int i;
    char *zone;

    for (i=0; env[i] != NULL; i++)
        printf("VARIABLE ENVIRONNEMENT:%s\n", env[i]);

    zone = (char *) malloc(TAILLE_ALLOC);
    if (argn != 2){
        printf("il faut 1 argument : le nb de pas de calcul\n");
        exit (-2);
    }

    sscanf(argv[1], "%d", &nb_pas);
    printf("\n\n\nnombre de pas = %d \n", nb_pas);
    printf("somme trigo = %d\n", calcul_trigo(nb_pas));
    printf("somme carre = %d\n", calcul_carre(nb_pas));

    printf("Adresse de main      = %X\n", main);
    printf("Adresse de TAILLE_ALLOC = %X\n", &TAILLE_ALLOC);
    printf("Adresse de nb_pas      = %X\n", &nb_pas);
    printf("Adresse de zone        = %X\n", zone);
    printf("Adresse de argv[1]     = %X\n", argv[1]);

    sleep(5);
    exit(nb_pas);
}

```

Ce programme, dont le code source est disponible en téléchargement depuis le site Pleiad, contient 4 sections :

- Les directives d'inclusion qui permettent d'utiliser des codes existants, contenus dans d'autres fichiers.
- Les variables globales, visibles dans l'ensemble du programme
- Les fonctions, unités d'exécution dans le langage C, chacune contenant des variables locales, visibles dans le corps du programme où elles sont définies
- Le programme principal qui fait appel aux différentes fonctions : le programme principal indique au système où commencer l'exécution du programme mais est structurellement une fonction ordinaire (excepté son nom : main)

#### 4.2.1 Compilation

La compilation du programme précédent est illustrée ci-dessous :

```
ael@ubuntu: ~/Documents/CNAM
File Edit View Search Terminal Help
ael@ubuntu:~/Documents/CNAM$ ls -l chap1*
-rwxr-xr-x 1 ael ael 1154 2012-02-27 14:56 chap1.c
ael@ubuntu:~/Documents/CNAM$ gcc -c chap1.c
ael@ubuntu:~/Documents/CNAM$ ls -l chap1*
-rwxr-xr-x 1 ael ael 1154 2012-02-27 14:56 chap1.c
-rw-r--r-- 1 ael ael 2096 2012-03-20 15:43 chap1.o
ael@ubuntu:~/Documents/CNAM$
```

1. fichier source
2. compilation avec le compilateur gcc
3. fichier objet créé

Le fichier ainsi créé se compose des parties suivantes :

- Text : l'appellation est trompeuse, il s'agit en fait de l'ensemble du code, instructions machines
- Data : données du programme ;
- Rodata : données en lecture seule (Read Only) ;
- Symbol : table des symboles ;
- Comment : informations sur le compilateur

Chaque fichier objet (ou exécutable) possède une table décrivant les différents symboles qu'il contient. La commande **nm** permet de visualiser la table des symboles, en fournissant : son nom, sa valeur, sa classe et son type.



```
ael@ubuntu: ~/Documents/CNAM
File Edit View Search Terminal Help
ael@ubuntu:~/Documents/CNAM$ nm chap1.o
00000000 D TAILLE_ALLOC
               U _isoc99_sscanf
00000059 T calcul_carre
00000000 T calcul_trigo
               U exit
0000008a T main
               U malloc
               U printf
               U puts
               U sin
               U sleep
ael@ubuntu:~/Documents/CNAM$
```

1. Le symbole est une variable ;
2. Le symbole est défini dans le code ;
3. Le symbole est indéfini.

Certains symboles ne sont donc pas "résolus". Il s'agit dans notre cas de fonctions définies dans d'autres bibliothèques.

Remarque : la sortie de la commande `nm` (ainsi que celle de la majorité des commandes présentées dans le cadre de ce cours) pourront présenter des variantes selon le système sur lequel elle est implémentée (Mac OSX, Linux, BSD, etc...)

#### 4.2.2 Edition de liens

L'édition de lien permet de constituer un fichier exécutable à partir de bibliothèques et de modules objets compilés séparément. L'éditeur de lien doit donc résoudre pour chacun des modules les références externes non résolues, en extrayant de bibliothèques les modules correspondant à des fonctions effectivement utilisées.

Historiquement l'édition de lien se faisait de manière *statique* : chaque code d'une fonction utilisée est recopié dans le fichier binaire. L'exécutable est alors autonome, mais si plusieurs programmes utilisent la même fonction, cette portion de code sera alors autant de fois chargée en mémoire centrale. De plus, la taille de l'exécutable augmente en conséquence. Le changement de version de bibliothèque ne se fera également pas sans souci.

Dans le cas de *l'édition de liens dynamique*, les références à résoudre vont contenir des informations sur les objets partagés qui seront chargés *si nécessaire* à l'exécution du programme.



```
ael@ubuntu: ~/Documents/CNAM
File Edit View Search Terminal Help
ael@ubuntu:~/Documents/CNAM$ gcc -w -static chap1.c -lm -o chap1-static
ael@ubuntu:~/Documents/CNAM$ gcc -w chap1.c -lm -o chap1-dynamic
ael@ubuntu:~/Documents/CNAM$ ls -l chap1*
-rwxr-xr-x 1 ael ael 1154 2012-02-27 14:56 chap1.c
-rwxr-xr-x 1 ael ael 7413 2012-03-20 15:51 chap1-dynamic
-rw-r--r-- 1 ael ael 2096 2012-03-20 15:43 chap1.o
-rwxr-xr-x 1 ael ael 616463 2012-03-20 15:51 chap1-static
ael@ubuntu:~/Documents/CNAM$
```

1. compilation avec édition de lien statique
2. compilation avec édition de lien dynamique

Comme attendu, le fichier exécutable résultant de l'édition de lien statique a une taille plus importante (616463 octets) que celui résultant de l'édition de lien dynamique (7413 octets).

Si l'on consulte la table des symboles du fichier chap1-dynamic, on pourra constater que tous les symboles sont maintenant résolus (extrait de la sortie de la commande nm) :

```
080485de T main
          U malloc@@GLIBC_2.0
          U printf@@GLIBC_2.0
          U puts@@GLIBC_2.0
          U sin@@GLIBC_2.0
          U sleep@@GLIBC_2.0
```

#### 4.2.3 Exécution

Une fois le fichier exécutable créé, il suffit sous Unix de taper la commande `./chap1-dynamic XX` (avec XX une valeur numérique) pour le lancer :

```
ael@ubuntu:~/Documents/CNAM$ ./chap1-dynamic 4
VARIABLE ENVIRONNEMENT:ORBIT_SOCKETDIR=/tmp/orbit-ael
VARIABLE ENVIRONNEMENT:SSH_AGENT_PID=1534
VARIABLE ENVIRONNEMENT:TERM=xterm
VARIABLE ENVIRONNEMENT:SHELL=/bin/bash
VARIABLE ENVIRONNEMENT:XDG_SESSION_COOKIE=0176efc2c3d9a942760bff2900000004-1331030411.746858-1146746660
VARIABLE ENVIRONNEMENT:WINDOWID=73418494
VARIABLE ENVIRONNEMENT:GNOME_KEYRING_CONTROL=/tmp/keyring-UHx3zX
VARIABLE ENVIRONNEMENT:GTK_MODULES=canberra-gtk-module
VARIABLE ENVIRONNEMENT:USER=ael
VARIABLE
ENVIRONNEMENT:LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tif=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01
```

```
;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx=
01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.mid=00;36:*.mi
di=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.
axa=00;36:*.oga=00;36:*.spx=00;36:*.xspf=00;36:
VARIABLE ENVIRONNEMENT:SSH_AUTH_SOCK=/tmp/keyring-UHx3zX/ssh
VARIABLE ENVIRONNEMENT:DEFAULTS_PATH=/usr/share/gconf/gnome.default.path
VARIABLE ENVIRONNEMENT:SESSION_MANAGER=local/ubuntu:@/tmp/.ICE-
unix/1504,unix/ubuntu:/tmp/.ICE-unix/1504
VARIABLE ENVIRONNEMENT:USERNAME=ael
VARIABLE ENVIRONNEMENT:XD_CONFIG_DIRS=/etc/xdg/xdg-gnome:/etc/xdg
VARIABLE ENVIRONNEMENT:DESKTOP_SESSION=gnome
VARIABLE
ENVIRONNEMENT:PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/u
sr/games
VARIABLE ENVIRONNEMENT:PWD=/home/ael/Documents/CNAM
VARIABLE ENVIRONNEMENT:GDM_KEYBOARD_LAYOUT=fr
VARIABLE ENVIRONNEMENT:LANG=en_US.UTF-8
VARIABLE ENVIRONNEMENT:GNOME_KEYRING_PID=1485
VARIABLE ENVIRONNEMENT:MANDATORY_PATH=/usr/share/gconf/gnome.mandatory.path
VARIABLE ENVIRONNEMENT:GDM_LANG=en_US.UTF-8
VARIABLE ENVIRONNEMENT:GDMSESSION=gnome
VARIABLE ENVIRONNEMENT:SHLVL=1
VARIABLE ENVIRONNEMENT:HOME=/home/ael
VARIABLE ENVIRONNEMENT:GNOME_DESKTOP_SESSION_ID=this-is-deprecated
VARIABLE ENVIRONNEMENT:LOGNAME=ael
VARIABLE
ENVIRONNEMENT:XD_DATA_DIRS=/usr/share/gnome:/usr/local/share:/usr/share/
VARIABLE ENVIRONNEMENT:DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-
b3EQ9jYRUq,guid=95c3b0f32be427037b163aa200000025
VARIABLE ENVIRONNEMENT:LESSOPEN=| /usr/bin/lesspipe %s
VARIABLE ENVIRONNEMENT:WINDOWPATH=7
VARIABLE ENVIRONNEMENT:DISPLAY=:0.0
VARIABLE ENVIRONNEMENT:LESSCLOSE=/usr/bin/lesspipe %s %s
VARIABLE ENVIRONNEMENT:XAUTHORITY=/var/run/gdm/auth-for-ael-6lkZLJ/database
VARIABLE ENVIRONNEMENT:COLORTERM=gnome-terminal
VARIABLE ENVIRONNEMENT:OLDPWD=/home/ael/Documents
VARIABLE ENVIRONNEMENT:_=./chap1-dynamic

nombre de pas = 4
somme trigo = 1
somme carre = 30
Adresse de main          = 80485DE
Adresse de TAILLE_ALLOC  = 804A02C
Adresse de nb_pas        = BF9CB5BC
Adresse de zone           = 778A1008
Adresse de argv[1]        = BF9CD635
ael@ubuntu:~/Documents/CNAM$
```

Le programme a simplement affiché les variables d'environnement de son contexte, et exécuté les deux fonctions. Pour son exécution, le système a chargé son code en mémoire, et exécuté les instructions sur le processeur, en mettant à jour les différents registres et le compteur ordinal.