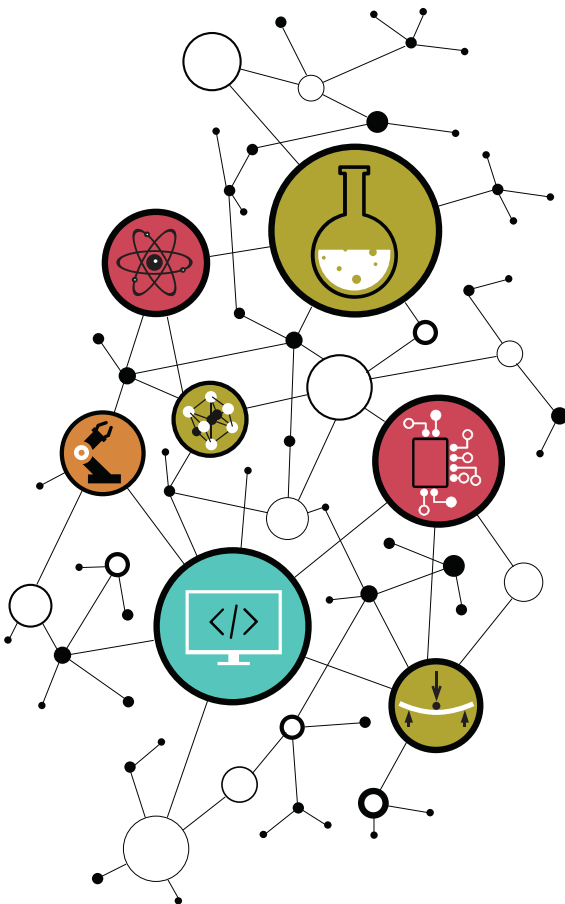


PARTIE 1

PRÉLUDE



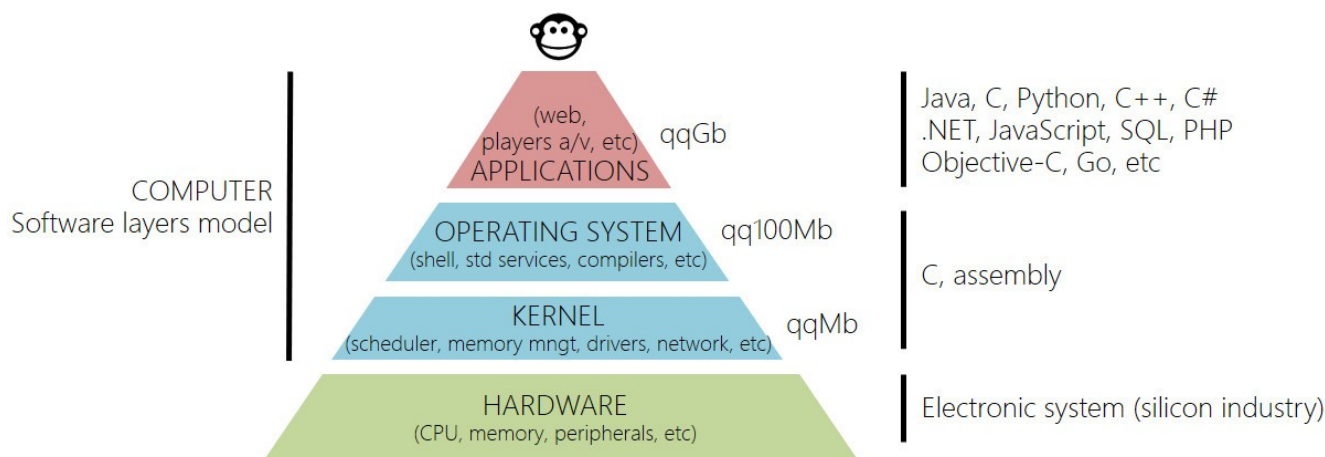
I. L'enseignement, en bref

Pour résumer ces exercices de travaux pratiques en une phrase, c'est de comprendre comment un programme initialement écrit en langage C fini par s'exécuter sur un ordinateur.

« Unix is basically a simple operating system,
but you have to be a genius to understand the simplicity. »

Dennis Ritchie, père du langage C et co-développeur d'Unix

Nous travaillons donc ici sur la partie applicative, dans ce qu'on appelle le **user-space**. Il s'agit de l'espace de la machine réservée à l'utilisateur (comme son nom l'indique), donc ce qui est laissé à disposition de n'importe quel programme. Par opposition et dans les couches logicielles plus basses, on trouve le **kernel-space** dans lequel le noyau du système d'exploitation travaille. C'est lui qui gère notamment les accès aux matériels de la machine. Cette deuxième partie sera étudiée au second semestre, dans l'**enseignement de Systèmes d'Exploitation et Réseaux**. Avec **Architecture des Ordinateurs**, ces deux enseignements complémentaires permettent d'appréhender une grande partie du fonctionnement des ordinateurs.



D'un point de vue **logiciel**, nous travaillerons avec les technologies et les standards les plus répandus à notre époque, dans le monde des couches basses des systèmes numériques de traitement de l'information : langage C (1972) ; système d'exploitation GNU (1983) ; chaîne de compilation GCC (C ANSI, 1986) ; interface standard POSIX (1988) ; noyau Linux (1991).

Ces standards, projets et technologies sont aux centres de la majorité des systèmes numériques d'information autour de nous de nos jours. Certains ont notamment inspiré et marqué l'émergence des concepts de logiciel libre et d'open source.

D'un point de vue **matériel**, les technologies choisies pour illustrer l'enseignement seront quant à elles celles restant toujours à notre époque les plus répandues sur ordinateur, soit les architectures compatibles x86/x64 (32-bit/64-bit). Ces technologies ont été historiquement développées et spécifiées par Intel (x86 IA-32 en 1985, compatibilité avec le processeur 8086 en 1978) puis AMD (x64 AMD64 compatible x86 en 2003), Intel étant la première société à avoir produit un microprocesseur (Intel 4004 en 1971).

II. Le langage C

Avant de présenter les objectifs de cet enseignement, quelques précisions concernant le langage C, son essence, son histoire et donc sa dominance dans le monde des couches basses des systèmes sont présentées à la suite ...

Le langage C fut historiquement développé par Dennis Ritchie (*AT&T, Bell Labs*) au début des années 70 afin de ré-écrire **Unix** (un système d'exploitation). Le langage C est une évolution du langage B (développé par Ken Thompson, père d'Unix). Rappelons qu'à notre époque les systèmes **Unix-like** (Linux, distributions GNU, MacOS, Android, ...) sont les plus répandus et toujours en pleine expansion sur les marchés. Il s'agit d'un langage de programmation impératif de bas niveau (proche de la machine). De nombreux langages plus modernes, mais néanmoins adaptés à d'autres usages, s'en sont inspirés (C++, Java, D, C#, PHP, JavaScript, ...). Même à notre époque, le C continue d'évoluer (normes C ANSI/C89 en 1989, C90, C95, C99, C11, C17 et dernièrement le C23).

Le langage C a pour intérêt d'avoir été pensé pour les couches basses des systèmes logiciels de supervision des machines numériques. Il offre un faible niveau d'abstraction au matériel et permet des analyses et des développements de plus bas niveau poussés et flexibles (assembleur, binaire). Il s'agit d'un langage "épuré" (comparé à d'autres comme le C++), versatile et permissif (la compilation et l'exécution tolèrent beaucoup de marge de manœuvre) offrant un contrôle important sur la machine, notamment au regard de la gestion mémoire (débordements, fuites, traces, ...). Il est donc à manier avec précaution et attention par le développeur. Vous en serez témoin à travers cette trame d'expérimentations. C'est d'ailleurs pour cela qu'il est toujours à notre époque utilisé pour développer les couches basses des systèmes (systèmes d'exploitation, systèmes embarqués, noyaux, compilateurs, interpréteurs, ...). Pour un développeur système, la complexité ne doit pas résider dans le langage mais dans le système !

À notre époque, il reste toujours parmi les langages les plus populaires au monde¹. Il est d'ailleurs en constante croissance à l'usage, notamment avec l'avènement des systèmes embarqués, de l'IoT (Internet des objets) et des projets « maker » (Raspberry Pi, Arduino, ...). Prenons des objets et logiciels de votre quotidien dans lesquels une partie voire la totalité des développements logiciels ont été développés en langage C : Box Internet, télévision, lecteurs CD/DVD/Blu-ray, enceinte Bluetooth, ordinateur et smartphone, systèmes d'exploitation (distributions GNU/Linux telles que Debian/Ubuntu/Redhat/Fedora/Kali, Android, MacOS X, iOS, Windows, ...) et noyaux (Linux, XNU, Darwin, Mach, NT, ...), la liste est très très très longue ...

1 <https://www.tiobe.com/tiobe-index/>

III. Objectifs pédagogiques

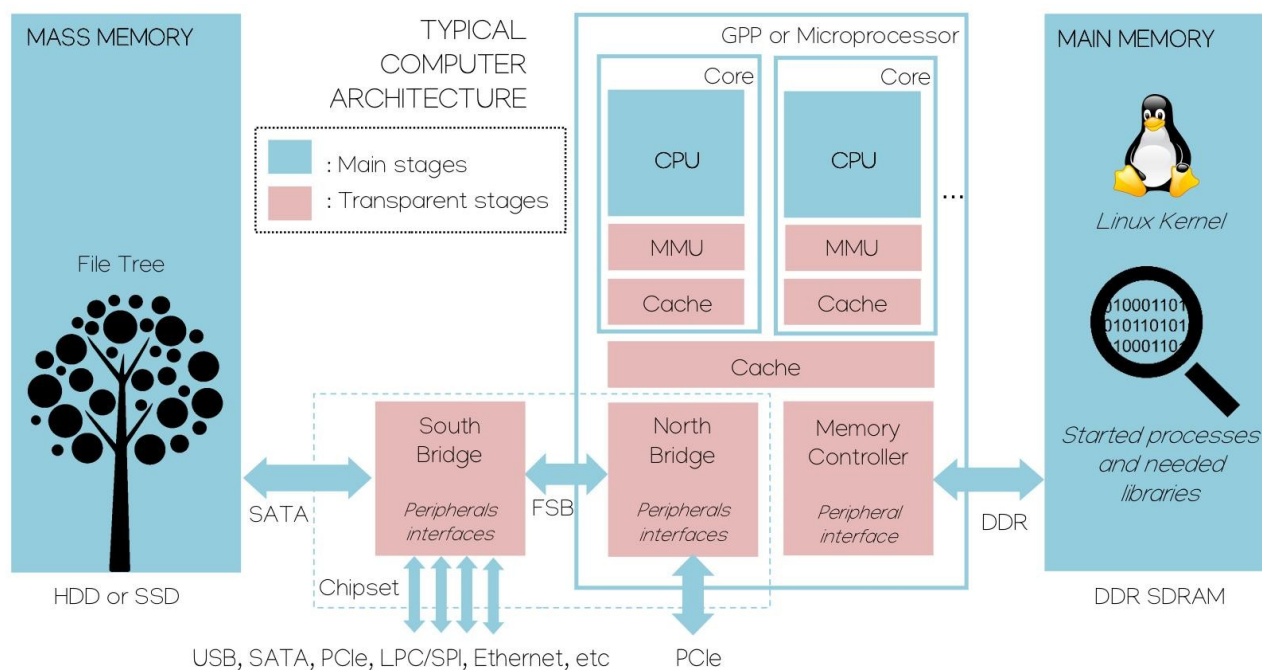
Cette trame d'enseignement ne cherche pas à vous ré-apprendre un langage de programmation. Il s'agit d'une trame d'analyse de programmes C élémentaires ayant pour objectif terminal d'aboutir à une meilleure représentation et compréhension du fonctionnement de la machine (ordinateur), de la compilation à l'exécution sur cible. L'objectif premier étant de mieux comprendre l'alchimie liant le système d'exploitation (*operating system*) au système d'exécution (hardware), mais également de maîtriser le processus de génération de micrologiciel (firmware) en partant d'un programme source (software).

Néanmoins, pour les plus clairvoyants, une bonne assimilation des connaissances et compétences enseignées, qu'elles soient architecturales ou techniques, pourrait bien changer à jamais votre façon de voir et d'écrire vos programmes à l'avenir. Ceci sera vrai, que vous deveniez développeur applicatif haut niveau (C++, Java, D, etc), tout comme développeur système bas niveau (C, assembleur).

Il ne s'agit donc pas d'une trame de travaux pratiques visant à apprendre à programmer. Néanmoins, nous nous attarderons sur des points que les enseignements d'initiation à la programmation en C passent très rapidement, car nécessitant une meilleure compréhension des couches basses du système. Prenons les exemples des qualificatifs de type et de fonction spéciaux, des classes de stockage, ... : `register`, `volatile`, `inline`, `static`, `const`, `restrict`, ...

III.1. Représentation physique de l'architecture matérielle

Les systèmes matériels actuels de traitement de l'information, tel qu'un ordinateur, sont devenus d'une grande complexité architecturale et technologique. À titre indicatif, à Caen chez NXP, le développement d'un simple composant sécurisé NFC (chiffrement de l'information) demande le travail direct de près de 700 ingénieurs. Les phases de cours seront là pour mieux comprendre les rôles des principaux éléments constitutifs de l'ordinateur. Nous nous attarderons sur les étages pouvant potentiellement un jour jouer un rôle sur les contre-performances de vos programmes (MMU et mémoires cache dans notre cas).

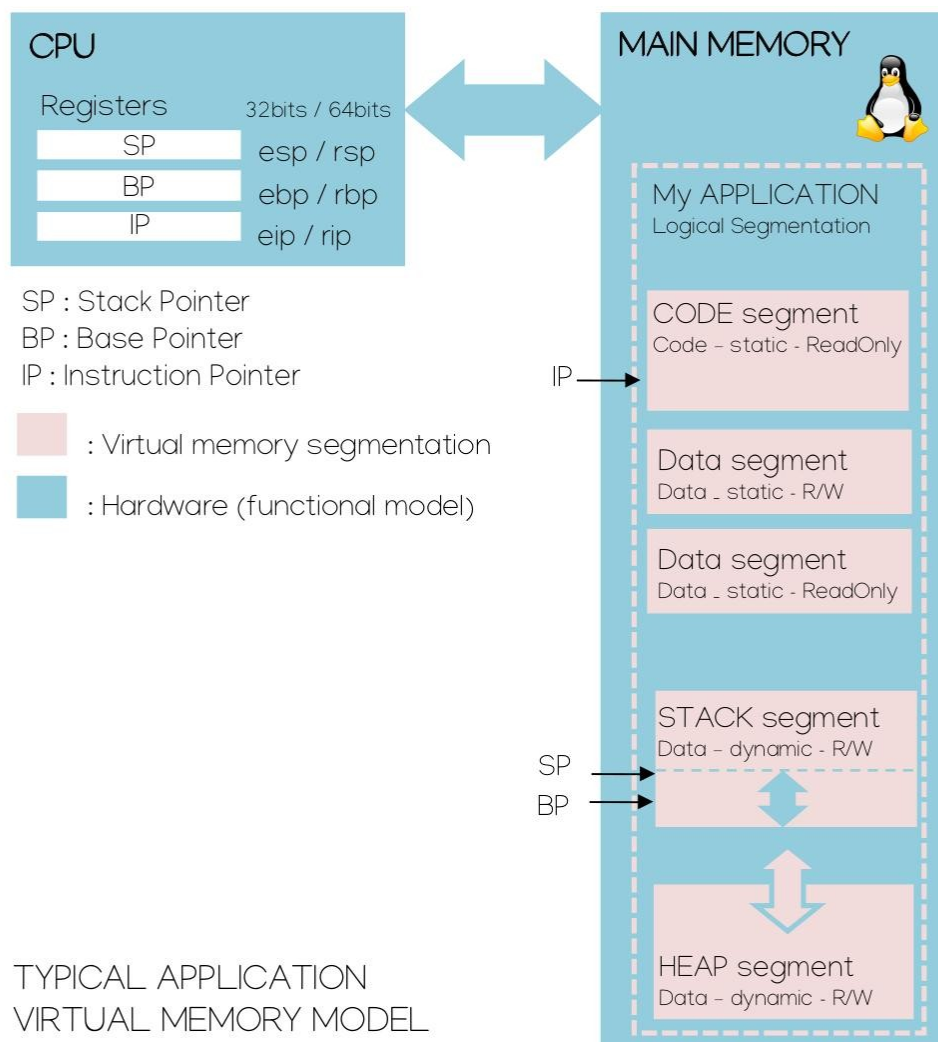


III.2. Représentation logique de l'architecture matérielle

Une fois l'environnement mémoire de l'application virtualisé, mappé et protégé (segmentation logique), puis son code et ses données statiques binaires chargés en mémoire principale depuis la mémoire de masse par le noyau du système d'exploitation (Linux par exemple), il ne reste alors plus qu'à exécuter l'application sur le CPU courant. Le modèle de plus bas niveau de représentation de la machine vu du CPU (étage registre) et du développeur (langage d'assemblage) reste à ce niveau relativement simple (cf. schéma ci-dessous). Cette machine minimale est souvent historiquement nommée machine de Von Neumann (mathématicien 1945, projet EDVAC). Il s'agit d'un modèle de machine à mémoire unifiée pour le stockage du code et des données (contrairement aux architectures de Harvard). Durant des phases de développement, une compréhension fine des contraintes amenées par cette segmentation logique représentant les limites environnementales en mémoire d'une application (frontières/périmètre), permet alors de garantir un réel durcissement d'un programme durant l'édition puis l'exécution.

La bonne compréhension des points précédemment cités ainsi que des deux schémas présentés (Architecture matérielle – représentations physique et logique) fait partie des attentes terminales de cet enseignement.

Il sera donc essentiel de revenir sur ces deux schémas vers la fin des enseignements, afin de vous assurer d'avoir compris ces deux figures.



IV. Outils et environnement de développement

L'archive de travail est directement téléchargeable sur la plateforme pédagogique de l'ENSICAEN :

- 2A GPSE SATE : <https://foad.ensicaen.fr/course/view.php?id=696>
- 2A GPSE FISA : <https://foad.ensicaen.fr/course/view.php?id=1022>

L'ensemble de la trame de Travaux Pratiques sera réalisée sur système GNU/Linux (Ubuntu dans notre cas, mais n'importe quelle distribution fera l'affaire). La compilation et l'édition des liens se feront donc sur une chaîne de compilation, outils de développement et ABI (*Application Binary Interface*) GNU GCC (GNU Collection Compiler)². Les logiciels GNU sont tous des logiciels libres et ouverts, soutenus par la *Free Software Foundation* (fondateur en 1985 Richard Stallman). Pour information, les chaînes de compilation GNU-like sont à notre époque les plus répandues, notamment dans le domaine des systèmes embarqués (supportant notamment les architectures Microchip, ARM, MIPS, Open Hardware RISC-V, ...). Elles s'imposent de plus en plus comme un standard.

Installation du système et des packages

Si vous souhaitez installer les outils sur votre machine personnelle et garder une configuration système proche des machines de l'école, nous vous conseillons d'installer un système Ubuntu LTS (*Long-Term Support*) réel ou virtualisé³. La version actuellement utilisée à l'école est Ubuntu 22.04 LTS. Néanmoins, même si cela est recommandé, il ne s'agit en rien d'une obligation, tant qu'un GCC est présent dans le système et que Linux supervise la machine hôte.

De même, si vous possédez déjà un système Windows sur votre machine, vous avez notamment la possibilité d'installer un Linux en *dual-boot* à côté de Windows ou de virtualiser Ubuntu sur une machine virtuelle. Le projet *VirtualBox*⁴ est par exemple un projet Open Source performant, bien maintenu et stable. Vous trouverez de nombreux tutoriels permettant d'installer un Ubuntu sur *VirtualBox*. Ne pas oublier de paramétrer votre configuration système sous *VirtualBox* en offrant les ressources suffisantes à votre système virtualisé à l'usage (virtualisation matérielle VT-x/AMD-V, 3/4 des ressources CPU, 3/4 des ressources RAM, ...) et en configurant le réseau.

Une fois votre système installé, voici potentiellement ci-dessous quelques packages complémentaires à installer.

```
sudo apt-get update
```

```
sudo apt-get install build-essential gcc-multilib binutils putty
```

² <http://gcc.gnu.org/>

³ <https://ubuntu.com/#download>

⁴ <https://www.virtualbox.org/wiki/Downloads>

V. Découpage temporel

Le document imprimé ne contient que les chapitres étudiés en TP.

Le document numérique contient l'intégralité des chapitres (dont ceux traités uniquement avec les INFO). Ces chapitres sont laissés à votre disposition pour compléter votre compréhension de l'enseignement et vous pouvez évidemment demander un support à vos enseignants pour ces parties hors TP.

Séances 1 et 2 :

Partie 1 Prélude.....	7
Partie 2 Chaîne de compilation.....	15

Séance 3 :

Partie 3 Allocation statique et fichier ELF.....	35
Partie 4 Assembleur Intel x86.....	45

Séance 4 et 5 :

Partie 5 Allocation automatique et segment de pile.....	57
---------------------------------------------------------	----

Séance 6 :

Partie 6 Allocation dynamique et segment de tas.....	75
Partie 7 Mémoires cache.....	83

Non-abordé en TP :

Partie 8 Exceptions matérielles et Signaux UNIX.....	89
------------------------------------------------------	----

Séance 7 :

Partie 9 Hacking.....	97
-----------------------	----

Non abordé en TP

Partie 10 Mémoire de masse et stockage des fichiers.....	109
----------------------------------------------------------	-----

