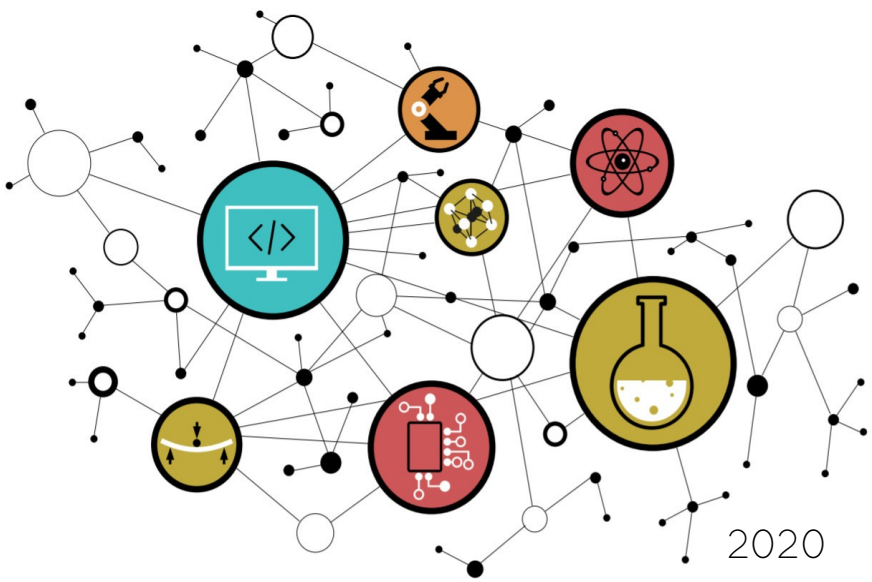
creative commons

TRAVAUX PRATIQUES

PRÉLUDE



2020

SOMMAIRE

La trame de TP minimale que nous considérons comme être les compétences minimales à acquérir afin d'accéder à aux métiers de base du domaine suit le séquencent suivant : chapitres 1, 2, 3, 4, 5 et 6 (voire 8 pour une mise en application et 9 pour la gestion des ressources de stockage de masse). Le reste de la trame ne sera pas évalué et représente une extension au jeu de compétences minimales relatif au domaine en cours d'étude (* devant chapitres facultatifs voire complémentaires). Libre à vous d'aller plus loin selon votre temps disponible et votre volonté de mieux comprendre et maîtriser ce domaine !

PRÉLUDE

1. PRÉLUDE

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

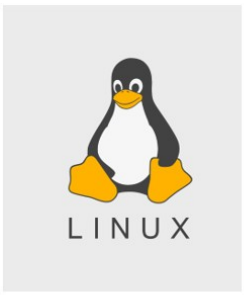
Dennis Ritchie, père du langage C et codéveloppeur d'Unix, à droite sur la photo ci-dessous

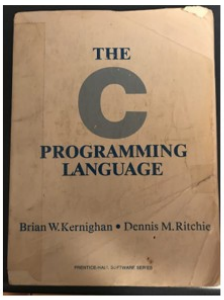
« I think the major good idea in Unix was its clean and simple interface: open, close, read, and write. »

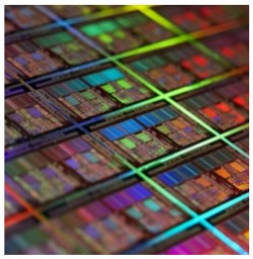
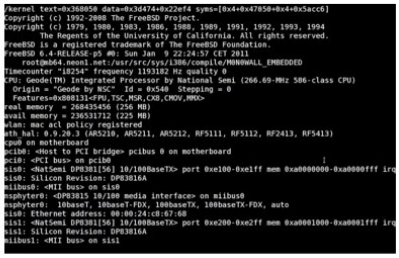
Kenneth Thompson, père du système d'exploitation Unix et du langage B, à gauche sur la photo

« Most of the good programmers do programming not because they expect to get paid or get adulation by the public, but because it is fun to program. »

Linus Torvalds, père de noyau Linux





Cet enseignement sera illustré sur les technologies et les standards logiciels les plus répandus à notre époque dans le monde des couches basses des systèmes numériques de traitement de l'information, notamment sur ordinateur (langages, systèmes d'exploitation, noyaux, compilateurs, systèmes embarqués, etc). Langage C (1972), système d'exploitation GNU (1983), chaîne de compilation GCC (C ANSI, 1986), interface standard POSIX (1988), noyau Unix-like Linux (1991), etc, la plupart de 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.

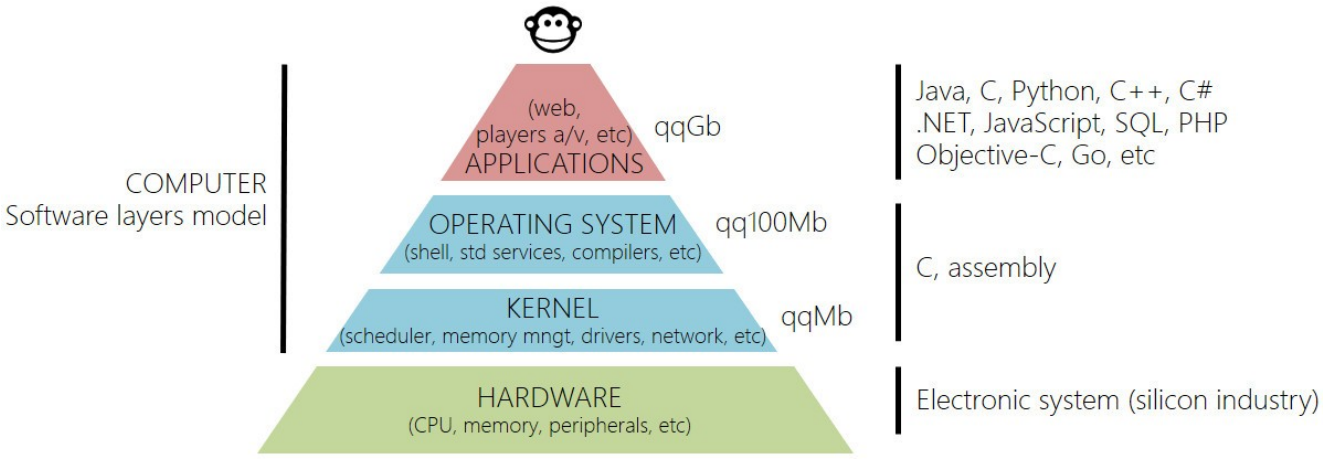
Les technologies matérielles choisies pour illustrer l'enseignement seront quant à elles celles restant toujours à notre époque les plus répandues sur ordinateur, soit les architectures compatibles x86/x64 (32bits/64bits). Ces technologies ont été historiquement développées et spécifiées par Intel (x86 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). 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 …

« C is declining somewhat in usage compared to C++, and maybe Java, but perhaps even more compared to higher-level scripting languages. It's still fairly strong for the basic system-type things. »

« The only way to learn a new programming language is by writing programs in it. »

Dennis Ritchie

Le langage C fut historiquement développé par Dennis Ritchie (AT&T, Bell Labs) au début des années 70's afin de réécrire 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, BSD, System V, etc) 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, etc). Même à notre époque, le C continu d'évoluer (normes C ANSI/C89 en 1989, C90, C95, C99, C11 et C18 en 2018, https://www.iso.org/standards.html).



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, etc). Il s'agit d'un langage ''épuré'' (comparé à d'autres comme le C++, D, etc), versatile et permissif (la compilation et l'exécution tolère beaucoup de marge de manœuvre) offrant un contrôle important sur la machine, notamment au regard de la gestion mémoire (débordements, fuites, traces, etc). 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, etc). Pour un développeur système, la complexité ne doit pas résider dans le langage mais dans le système !

A notre époque, il reste toujours parmi les langages les plus utilisés au monde (n°1 en 2020, source TIOBE index, https://www.tiobe.com/tiobe-index/). 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, etc). 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/etc, Android, Mac OS X, iOS, Windows, etc) et noyaux (Linux, XNU, Darwin, Mach, NT, etc)), la liste est très très très longue ...

1.1. Objectifs pédagogiques

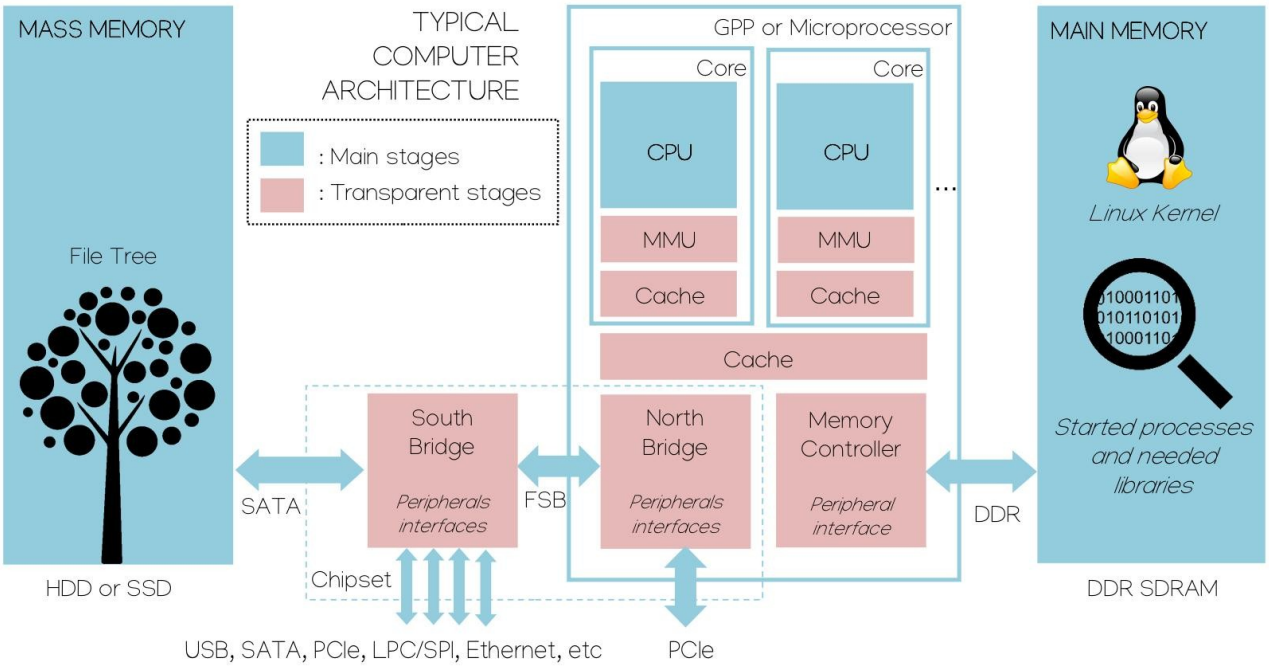
Cette trame d'enseignement ne cherche pas à vous réapprendre un langage de programmation. Il s'agit d'une trame d'analyse de programmes C élémentaires ayant pour objectif terminal d'aboutir à une meilleure représentation et compréhension du fonctionnement de la machine (ordinateur), de la compilation 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 souvent très rapidement, car nécessitant une meilleure compréhension des couches basses du système. Prenons les exemples des qualificateurs de type et de fonction spéciaux, des classes de stockage, etc (register, volatile, inline, static , const, restrict, etc).

Architecture matérielle (représentation physique)

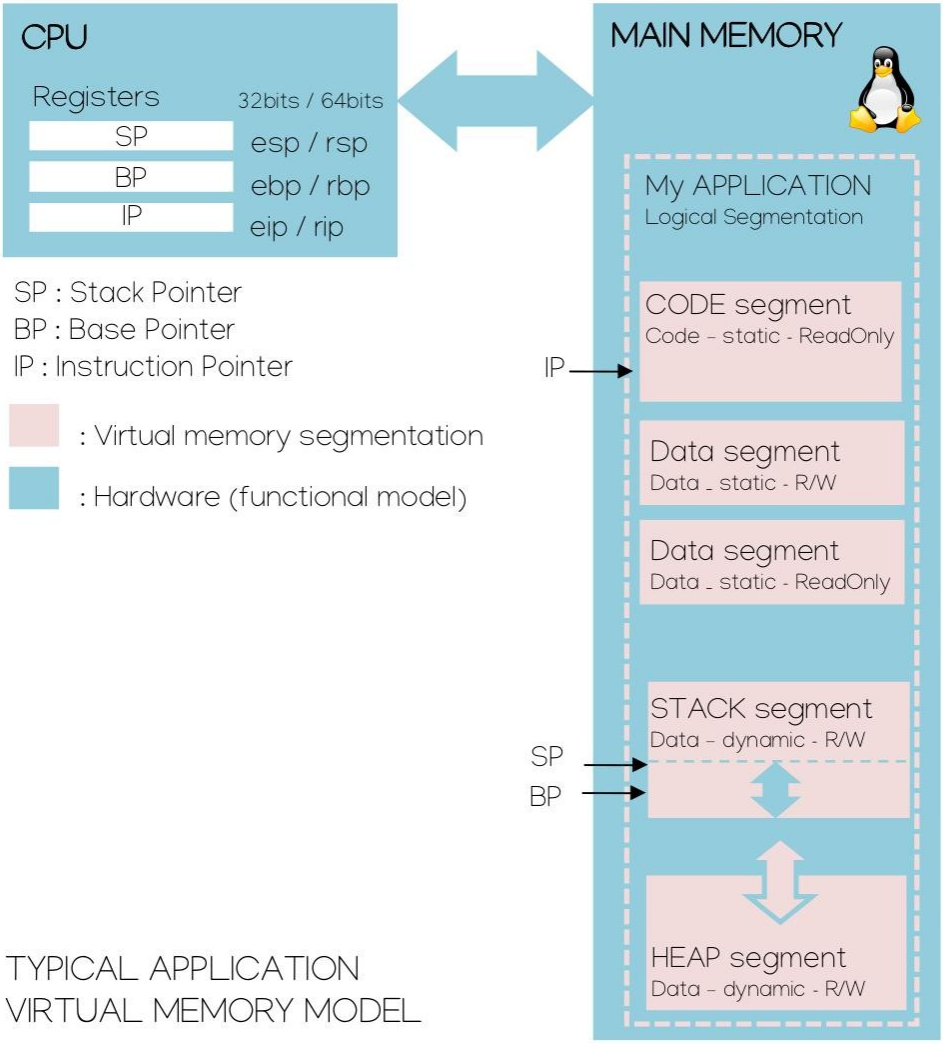
Les systèmes matériels actuels de traitement de l'information, tel qu'un ordinateur, sont devenus d'une grande complexité architecturale et technologique. A titre indicatif, à Caen chez NXP, le développement d'un simple composant sécurisé NFC (chiffrement de l'information) demande le travail direct de 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 contres performances de vos programmes (cache processeur, MMU, etc).



Architecture matérielle (représentation logique)
Environnement mémoire segmenté d'une application

Une fois l'environnement mémoire de l'application virtualisé, mappé et protégé (segmentation logique), puis son code et ses données statiques binaires chargées 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.



1.2. Outils et environnement de développement

« Sharing knowledge is the most fundamental act of friendship. Because it is a way you can give something without loosing something. »

Richard Stallman, créateur du projet GNU et fondateur de la Free Software Foundation

L'archive de travail est directement téléchargeable sur la plateforme pédagogique de l'ENSICAEN (https://foad.ensicaen.fr) : *Formation Classique > Spécialité Informatique > 1ère année > Architectures des ordinateurs > Download > «année courant» > arch.zip et disco.zip*. Les programmes à analyser se situent dans le répertoire *arch/tp/disco* (disco pour discovery).

L'ensemble de la trame de Travaux Pratiques sera réalisée sur système GNU\Linux (Ubuntu, Debian, Fedora, Mageia, etc la liste est longue). 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, http://gcc.gnu.org/). Les logiciels GNU sont tous des logiciels libres et ouverts, soutenus par la Free Software Foundation (fondateur en 1985 Richard Stallman, https://stallman.org/). Pour en savoir plus sur la philosophie de ces projets voire y collaborer, n'hésitez pas à vous rendre sur les sites officiels (https://www.gnu.org/software/). Pour information, les chaînes de compilation GNU-like sont à notre époque les plus répandues, notamment dans le domaine des systèmes embarqués (architectures supportées ARM, MIPS, Open Hardware RISC-V, STMicro, Microchip, etc). Elles s'imposent de plus en plus comme un standard.

```
wget https://foad.ensicaen.fr/<moodle_path>/arch.zip

unzip arch.zip -d <destination_folder>
```

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 virtualisé (https://ubuntu.com/#download). 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 (https://doc.ubuntu-fr.org/cohabitation_ubuntu_windows) ou de virtualiser Ubuntu sur une machine virtuelle. Le projet VirtualBox est par exemple un projet Open Source performant, bien maintenu et stable (https://www.virtualbox.org/wiki/Downloads). 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, etc) et en configurant le réseau.

Une fois votre système installé, voici potentiellement ci-dessous quelques packages complémentaires à installer. A partir de maintenant, la console système (shell), un éditeur de texte (emacs, vim, nano, geany, etc), un compilateur GNU (gcc), un noyau Linux, votre ordinateur et votre volonté de comprendre un peu mieux ce monde obscure du système seront vos principaux alliés ...

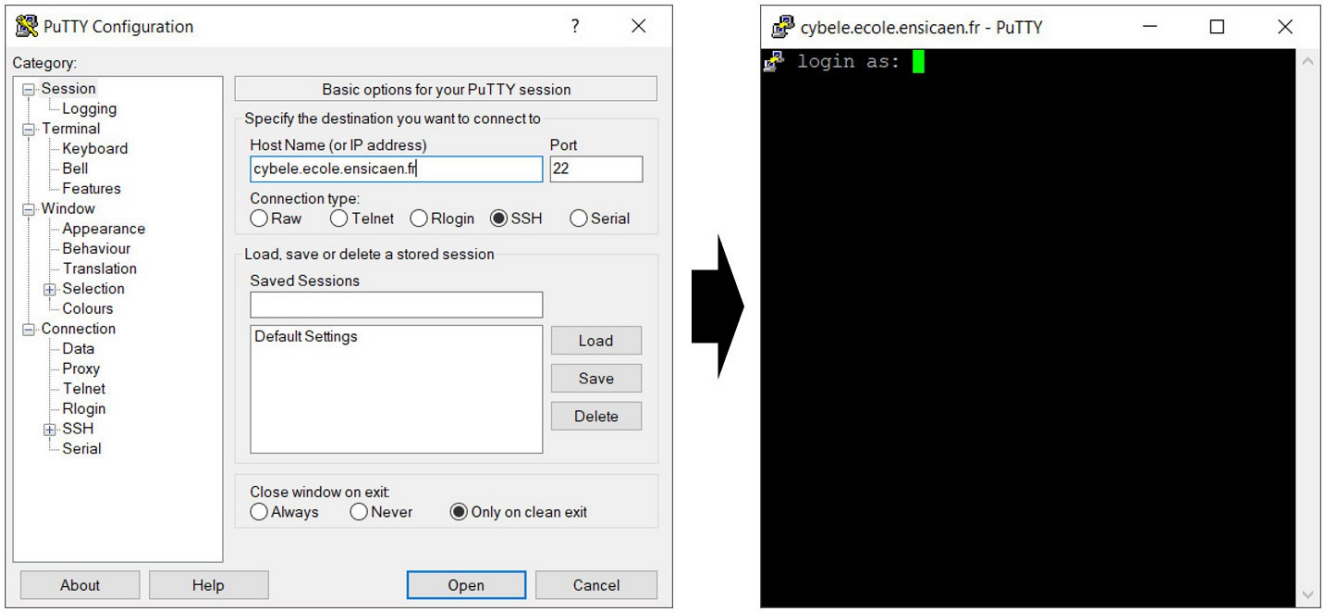
```
sudo apt-get update

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

Travail à distance

Pour divers raisons, si vous ne souhaitez ou ne pouvez pas installer un système GNU\Linux sur votre machine personnelle, il vous est également possible de travailler à distance depuis n'importe quelle machine (Windows, Linux, etc) possédant le logiciel PuTTY (ou un autre terminal SSH). PuTTY est un programme léger pouvant notamment vous offrir un terminal et lien de communication sécurisé SSH (Secure Shell) distant avec une autre machine. Dans notre cas, nous l'utiliserons pour nous connecter à notre compte ENSICAEN directement sur le serveur école. Voici le lien officiel de téléchargement :

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

La configuration de l'outil PuTTY est très simple (Host Name : *cybele.ecole.ensicaen.fr*, Port : *22*, Connection type : *SSH*, *Open* puis login et mdp : *ceux de zimbra*). Un raccourci pratique à connaître sous PuTTY est le clic droit de la souris permettant de coller une sélection :



Archive de travail pratique et édition en mode console

L'archive *disco.zip* ne contient que les fichiers textes sources de TP à analyser (archive légère de qq10Ko, aucun support PDF ni ODT de CM ni de TP). Une copie de l'archive est présente sur le serveur cybele ENSICAEN au chemin suivant : */home/public/arch/disco.zip*. Nous aurons à utiliser un éditeur de texte en mode console. Nous utiliserons *nano* également porté par le projet GNU (https://www.nano-editor.org/), éditeur le plus simple de sa famille (vi, vim, etc). Lien vers la totalité des raccourcis *nano* : https://www.nano-editor.org/dist/latest/cheatsheet.html

```
cd <working_directory>

cp /home/public/arch/disco.zip .

unzip disco.zip

rm disco.zip
```

1.3. Séquencement pédagogique

Le plan et le séquencement de la trame de Travaux Pratiques correspondent au chemin proposé afin d'atteindre les objectifs pédagogiques fixés. Ces objectifs ont été choisis au regard des attentes et exigences demandées par les marchés de l'industrie du logiciel et des couches basses des systèmes : systèmes embarqués, systèmes temps réel, développement de systèmes d'exploitation, développement de bibliothèques spécialisées, développement de chaînes de compilation, attaque et sécurité des systèmes, etc, tous des métiers dans divers domaines actuellement exercés par certains de nos anciens élèves. Il s'agit d'un séquencement conseillé qui n'a en aucune façon volonté à être imposé (étudiant comme enseignant encadrant). Pour se dérouler sous les meilleurs hospices, il serait préférable dès le début de l'enseignement de rythmer 1 à 2 heures de travail personnel à la maison en dehors des séances en présentiels avec enseignant. Voici une proposition de séquencement. Lire à minima l'introduction de chaque TP avant de venir en séance :

2. Compilation et éditions de liens

- En session de TP : *2.5 / 2.6 / 2.7 (TP n°1)*
- Hors session de TP : *Lire le document prélude. Rédiger une synthèse de 3 pages sur la compilation et l'édition des liens à déposer sur moodle (avant TP n°1) et 3.1 (avant TP n°2)*

3. Assembleur x86 et bibliothèque statique

- En session de TP : *3.1 / 3.2 / 3.3 / 3.4 / 3.5 / 3.6 (TP n°2)*
- Hors session de TP : *4.1 (avant TP n°3)*

4. Allocations automatiques et segment de pile

- En session de TP : *4.1 / 4.2 (TP n°3) et 4.3 / 4.4 (TP n°4) et 4.5 / 4.6 (TP n°5)*
- Hors session de TP : *4.3 (avant TP n°4) et 4.5 (avant TP n°5) et 5.1 (avant TP n°6)*

5. Allocations statiques et fichier ELF

- En session de TP : *5.1 / 5.2 / 5.3 (TP n°6)*
- Hors session de TP : *6.1 (avant TP n°7)*

6. Allocations dynamiques et segment de tas

- En session de TP : *6.1 / 6.2 (TP n°7)*

7. Exceptions et signaux Unix

- *Facultatif*

8. Hacking

- En session de TP : *8.1 / 8.2 / 8.3 (TP n°7)*
- Hors session de TP : *9.1 (avant TP n°8)*

9. Mémoire de masse et système de fichiers

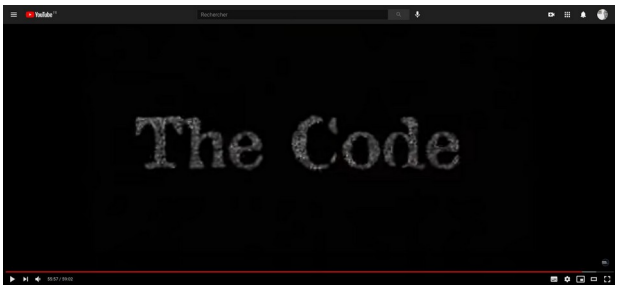
- En session de TP : *9.1 / 9.2 / 9.3 (TP n°8)*

1.4. Quelques ressources internet

Voici quelques ressources en ligne pouvant vous aider à une meilleure contextualisation des machines, des outils de développement, des langages et des systèmes utilisées à notre époque. Bons visionnages et lectures dans ces voyages dans notre histoire …

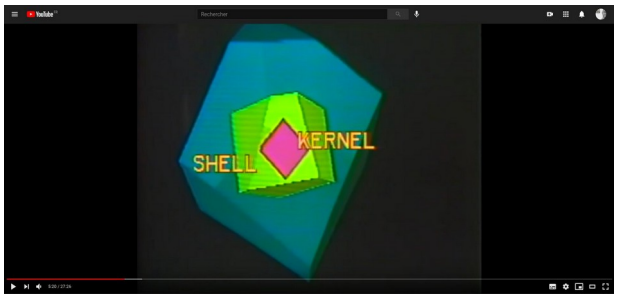
https://www.youtube.com/watch?v=dcN9QXxmRqk

Noyau Linux et système d'exploitation GNU – Nom de code Linux



https://www.youtube.com/watch?v=79_IMeks4wY

Système d'exploitation Unix – AT&T Archives: The Unix Operating System



https://www.youtube.com/watch?v=XvDZLjaCJuw

Sociétés privées Apple, Microsoft et IBM – Les cinglés de l'informatique

https://www.youtube.com/watch?v=dOakyAhqiVY&t=2140s

https://www.youtube.com/watch?v=zywPwbQqshY&t=2364s

https://www.youtube.com/watch?v=m9QUs2Nf3yA