**Implémentation d’un interpréteur de commande**

**Rapport**

*Anizet Thomas Lekens Amaury Vandenbusche Benjamin*

*[14164@ecam.be](mailto:14164@ecam.be) [14000@ecam.be](mailto:14000@ecam.be) [13152@ecam.be](mailto:14000@ecam.be)*

**22 Décembre 2017**

Intitulé du cours : Laboratoire de système d’exploitation

Dates du laboratoire : 3 séances équitablement réparties

# 1. Introduction

Dans le cadre du cours de *Système d’Exploitation* vu en 1ère année Master à l’ECAM, il a été demandé aux étudiants d’implémenter un interpréteur de commandes en C. Diverses consignes leurs ont été imposées selon une grille d’évaluation fichée en annexe.

# 2. Domaine d’application et commandes

Le domaine d’application concerne un robot autonome tout-terrain avec caméra et GPS embarqués. Ce projet robot étant réalisé dans une autre unité de l’enseignement, il nous a semblé judicieux d’implémenter 3 commandes qui nous permettraient de tester rapidement et aisément le bon état de fonctionnement de certains paramètres du robot. Ainsi, les X commandes choisies sont les suivantes :

* Commande *Forward* : …
* Commande setPin: Commande qu’on utilise pour spécifier le numéro des pins utilisées par le driver. Une fois lancé elle demande à l’utilisateur le numéro de chaque pin un à un. Elle crée ensuite une liste de quatre chiffres et ensuite elle sauvegarde ces quatres chiffres dans un simple fichier txt (PinData.txt). Ce fichier-ci sera utilisé par d’autres fonctions (comme goForward par exemple) pour décider l’état des pins.
* Commande ***GPS***: Cette commande ne prend aucun paramètre en entrée. Elle renvoie simplement les coordonnées (longitude, latitude) de la position actuelle du robot. 3 erreurs sont possibles :
  + Ouverture impossible du port série (c’est le cas si nous lançons la commande *GPS* et que le port série est déjà utilisé par une autre commande nécessitant le port série) ;
  + Aucune donnée n’est renvoyée (c’est le cas si le GPS n’a pu se connecté aux différents satellites) ;
  + Autres types d’erreur liés à la lecture du port série (fonction *Read()*) auquel cas, *errno* recensera le problème.

# 3. Appels systèmes et services offerts par l’OS

Un appel système est une fonction fournie par le *kernel* d’un système d’exploitation et utilisée par les programmes s’exécutant dans l’espace utilisateur. Un gros avantage des appels systèmes est que l’on assure une certaine sécurité des applications dans l’espace utilisateur.

Les appels systèmes disponibles dépendent du système d’exploitation utilisé mais plusieurs appels sont communs à la plupart des OS. Dans notre cas, nous utilisons Debian (GNU/LINUX) comme système d’exploitation.

Nous distinguons 6 catégories principales d’appels systèmes : Contrôle des processus, gestion des fichiers, gestion des périphériques, maintenance de l’informatique, communication, protection.

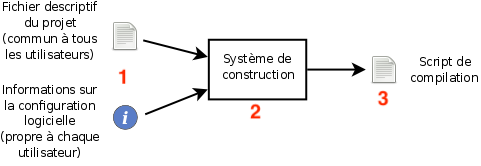
Dans notre cas, nous avons utilisé comme appels systèmes (selon catégories) :

* Pour la gestion des fichiers : *open*, *close*, *write*, *read*, …
* Pour le contrôle des processus : *pid\_t fork*, …
* … ?

# 4. Outil de compilation

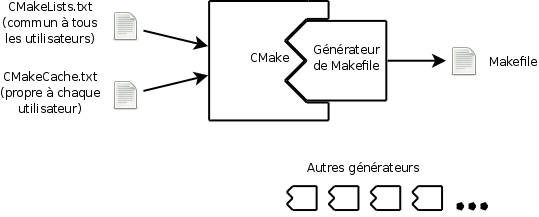
Les systèmes de construction logicielle ont pour but de permettre, directement ou indirectement, la compilation d'un code source. Tous les systèmes de construction (GNU Make, Ant, SCons, Waf, CMake, Meson, …) ont leurs spécificités, mais leur fonctionnement est globalement le même, à savoir :

1. Le développeur écrit dans un fichier la description du projet (incluant la liste des fichiers sources, les bibliothèques à lier au binaire final, etc.).
2. Les personnes souhaitant compiler le projet appellent le système de construction en lui passant en paramètre le fichier écrit par le développeur. Le système analyse alors le fichier d'une part et tente de récupérer des informations sur la configuration logicielle de la machine (fichier créé automatiquement) l'exécutant d'autre part.
3. Une fois ces opérations accomplies, il sera enfin en mesure de produire un script de compilation de bas niveau.



*Figure 1* – Fonctionnement global d’un système de construction logicielle.

Dans notre cas, nous avons utilisé **CMake** comme système de construction logicielle. À partir du fichier descriptif du projet (nommé CMakeLists.txt) et des informations sur la configuration logicielle de la machine effectuant la compilation (listées dans le fichier CMakeCache.txt, pré rempli lors de la phase d'analyse de la configuration), CMake est capable de générer le fichier suivant : Makefile.



*Figure 2* – Fonctionnement de CMake.

Nous avons choisi CMake pour diverses raisons :

* CMake gère (entre-autre) le langage de programmation demandé, à savoir le C ;
* CMake est très facile à utiliser (ce qui est loin d'être le cas de tous les outils de cette catégorie) ;
* Son indépendance vis-à-vis du système d'exploitation, du compilateur et des outils de développement ;

# 5. Questions-Réponses

Il a également été demandé aux étudiants de répondre à 2 questions, à savoir :

### Un interpréteur de commandes peut exécuter les commandes lui-même ou déléguer cette exécution à d’autres exécutables. Quels sont les avantages et inconvénients de ces deux possibilités ?

**Avantages**:

* Plusieurs exécutables : créer une librairie de commandes qu’on peut utiliser autre part.
* Code plus structuré et lisible, diminue le couplage et augmente la modularité

**Inconvénients :**

* Légèrement plus difficile à implémenter
* … ?

### Pour interagir avec le hardware et le système d’exploitation, on peut utiliser un appel système, une fonction de la librairie standard C ou passer par un programme système. Quels sont les avantages et inconvénients de ces trois possibilités ?

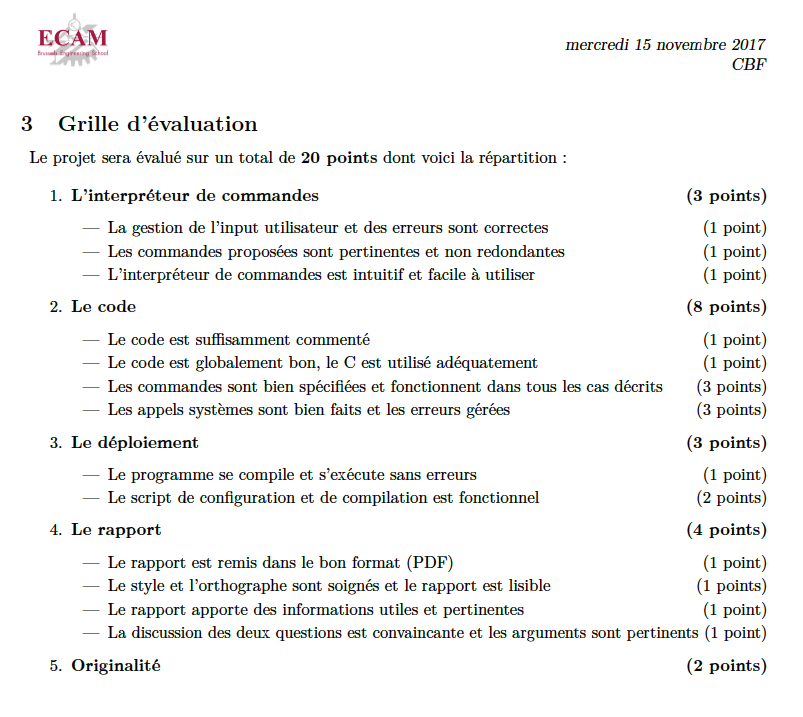
**Avantages**:

* Appel système : Optimisé pour un OS, (Atomicité ?)
* Librairie C : Peut fonctionner sur plusieurs OS différents
* Appel Système : possible de trouver quelque chose de plus spécifique / complexe qu’un appel système

**Inconvénients**:

* Appel système :
* Librairie C :
* Appel système :

**Annexe**:



***Annexe 1*** – Grille d’évaluation