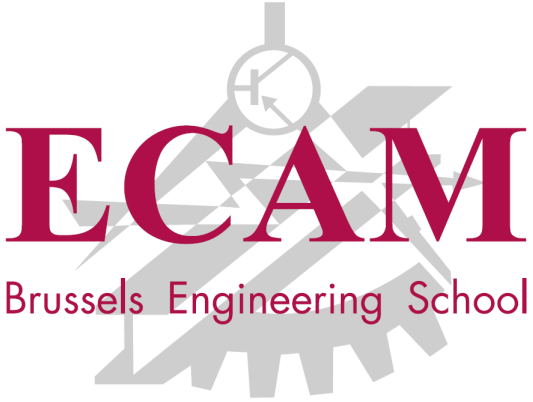
Laboratoire de système d'exploitation

Création d'un shell

17/12/2017

ALBERT Emile 14022@ecam.be
HAGOPIAN Armen 14040@ecam.be
SELLESLAGH Tom 14164@ecam.be



1 Introduction

Les objectifs de ce laboratoire étaient de penser et d'implémenter un shell. Notre groupe étant composé de trois électroniciens, tout trois participant au projet Eurobot, nous avons décidé de profité de ce laboratoire pour créer une interface système permettant nous permettant de monitorer les tests sur un robot commandé par une raspberry pi.

Le but de ce shell est donc de pouvoir observer lors de l'exécution d'un test de commande moteur sur la raspberry, un maximum d'information intéressante sur l'état du contrôleur. L'interface système devient alors une aide au debuging des programmes s'exécutant sur la raspberry.

2 Implémentation

2.1 L'environnement du shell

La structure du shell tel que nous l'avons utilisée dans ce projet a été repris du code crée par Stephen Brennan sur son compte Github : `https://github.com/brenns10/lsh`

La structure du shell peut être vue en plusieurs sections :

- La déclaration des fonctions de l'interface, des pointeurs vers ces fonctions et des noms d'appel
- L'implémentation des fonctionnalités
- Le cœur du shell

2.1.1 Déclaration des fonctions de l'interface, des pointeurs et des noms d'appel

Dans cette partie de configuration, il s'agit de déclarer les variables associées aux fonctions utilisées dans le programme.

pointeur ?

De plus, on définis les noms d'appel des fonctions. Ces nom seront ceux utilisé par l'utilisateur pour invoquer telle ou telle fonction

2.1.2 Implémentation des fonctions

Cette partie regroupe l'ensemble des fonctions appelée lors d'une commande utilisateur. Les fonctions propres à notre projet sont donc codée dans cette section ainsi que les commandes propres au shell (`help, clear, exit`)

2.1.3 Coeur du shell

Enfin, la dernière partie de la structure est composée des 6 fonctions requises au bon fonctionnement du shell. Ces 6 fonctions sont :

- `rpi_launch` - Lancement du shell
- `rpi_execute` -
- `rpi_read_line` - Lecture des commandes user
- `rpi_split_line` - Décryptage des commandes user
- `rpi_loop` - Super loop d'exécution
- `main` - Fonction principal du programme

2.2 Les commandes implémentées

2.2.1 `rpi_raspInfo` invoqué avec `rasp`

La commande `rasp` permet de récupérer des informations sur le hardware de la raspberry. Ces données peuvent être utiles tant pour le *debuging* que pour l'optimisation des performances d'un programme.

- `rasp -t -c ou -f` permet de récupérer la température du processeur en degré celsius ou farenheit
- `rasp -cpu` permet de récupérer l'utilisation du CPU au temps t
- `rasp -ram` permet de connaitre l'utilisation de la mémoire RAM

Appels systèmes
Un appel système est une fonction primitive fournie par le noyau d'un système d'exploitation et utilisée par les programmes s'exécutant dans l'espace utilisateur (en d'autres termes, tous les processus distincts du noyau). Ce système permet de contrôler de façon sécurisée les applications dans l'espace utilisateur.

2.3 Script de compilation

La compilation du code source écrit en C s'effectue sur la raspberry même. La commande `gcc` permet de créer un fichier ELF, exécutable sur le contrôleur.

3 Discussion

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 ?

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 ?