Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Alan Sueur
Nicolas Fuchs
Classe I-2a
Systèmes Embarqués
13.03.17

TP10b - Mini-Projet 2 - OS

Introduction

L'objectif de ce projet est de créer un système d'exploitation Linux en ligne de commande. L'exécution de notre projet se réalise sur la cible Black Beaglebone et permettra l'utilisation des différentes commandes suivantes :

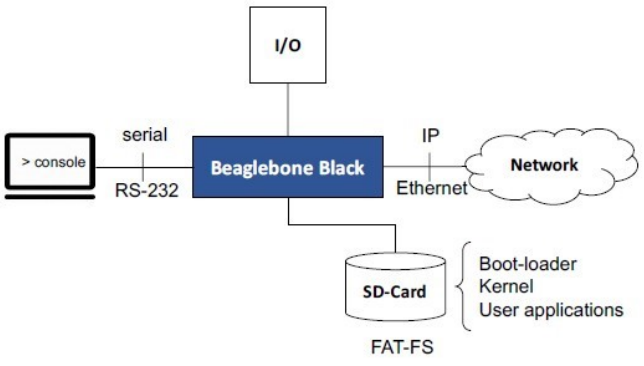
- cat : Visualisation du contenu d'un fichier
- cd : Changer de répertoire
- mkdir : Création d'un répertoire
- echo : Affichage d'un string avec redirection dans un fichier
- ps : Liste des processus et threads
- pwd : Affichage du répertoire courant (Cette fonctionnalité sera implémentée au travers du command prompt)
- rm : Effacement d'un fichier et/ou d'un répertoire
- fork : Lancement d'une nouvelle application stockée sur la SD-Card
- help : Affichage d'un texte d'aide générale ou pour une commande
- ls : Liste des fichiers contenus dans un répertoire

Une historique des commandes correctes à également été mise en place.

Voilà des exemples d'utilisation de ces commandes :

- cat file_name (on doit se trouver dans le bon répertoire)
- cd directory (un seul répertoire à la fois)
- mkdir directory (un seul répertoire à la fois)
- rm file/directory (le répertoire doit être vide)
- echo text_to_display / echo file < text_to_write (on peut écrire une chaîne de caractères avec espaces dans un fichier)
- ps
- pwd n'est pas implémenté en tant que commande mais est visible au travers du command prompt
- fork old_thread_name new_thread_name
- help / help command_name
- ls

Le projet sera divisé en deux parties : Une partie noyau qui permettra principalement de gérer la gestion des threads et des processus, gérer la MMU et accéder aux pilotes. La deuxième partie est l'application utilisateur qui permettra à celui-ci d'intéragir avec le noyau.



Alan Sueur
Nicolas Fuchs
Classe I-2a

Heures de travail

En dehors des heures de cours, nous avons du travailler environ 34h00. Durant le cours nous avons pu réaliser les tâches suivantes :

— Copie et compréhension du code mis à disposition par Gachet [15h00]

A la maison, nous avons du finir les tâches suivantes :

— Implémentation des fonctionnalités demandées (console, shell, exécution du buzzer, ...) [20h00]
— Implémentation du virtual file system [8h00]
— Tester le code et debugger [2h00]
— Etudier comment fonctionne le code et l'ensemble du projet [∼2h30]
— Rédiger ce rapport. [1h00]

Synthèse

Non acquis

— Ecriture des Makefiles

Partiellement acquis

— Polymorphisme (virtuel vs SD)
— Gestion de la MMU

Acquis

— Interruptions
— Gestion des threads
— Différences mode kernel/user

Remarques & Feedback

Ce projet a été un travail certes conséquent mais très intéressant. En aucun cas nous n'aurions pensé qu'il était possible de réaliser un système d'exploitation et encore moins en deuxième année. Cela nous a permis d'acquérir une expérience unique dans ce domaine, une expérience dont nous allons nous rappeler durant notre vie probablement.

Globalement, nous pensons que le niveau de ce projet était un peu trop élevé pour nous, sûrement dû au fait que la quantité de travail avec les autres cours nous limite dans notre investissement. C'est pourquoi nous avons du demander de l'aide à notre entourage, spécialement Monsieur Pont, afin de mieux comprendre certains concepts réalisés et savoir comment continuer. Cependant, avec les explications de différentes personnes, nous sommes parvenus à mieux comprendre le fonctionnement du projet.

Nous n'avons malheureusement pas eu assez de temps pour implémenter la partie user. Il aurait fallu dans le fichier syscall.c, faire des appels aux fonctions comme kernel_open, etc. Ces fonctions seraient directement appelées lorsque l'on se trouve en mode kernel et seraient appelées indirectement par des interruptions en mode user. Par après, dans syscall.c qui est accédé par les interruptions (bbb_syscall.c du mode user), ce sont ces mêmes fonctions (kernel_xxx) qui sont appelées. Les interruptions sont par contre utilisées quand même pour les fonctions kernel_thread_yield et kernel_thread_exit.

Malgré les parties compliquées du projet, nous avons bien aimé pouvoir réaliser un travail de cette envergure.