António Domingos Ana Sofia

**Cahier de bord**

Argentin Yvann

Hottinger Jeremy

Lopes Marques Vasco

Table des matières

[Etape 1 4](#_Toc403547684)

[Description détaillé 4](#_Toc403547685)

[Rôles 4](#_Toc403547686)

[Semaine 1 4](#_Toc403547687)

[Recherches : 4](#_Toc403547688)

[Travail effectué : 4](#_Toc403547689)

[Bilan visite professeur : 4](#_Toc403547690)

[Semaine 2 5](#_Toc403547691)

[Travail effectué : 5](#_Toc403547692)

[Bilan visite professeur : 5](#_Toc403547693)

[Etape 2 6](#_Toc403547694)

[Description détaillée 6](#_Toc403547695)

[Rôles 6](#_Toc403547696)

[Semaine 1 6](#_Toc403547697)

[Modification bootloader et stack 6](#_Toc403547698)

[Kernel en C, interruption et appels systèmes 6](#_Toc403547699)

[Fonctions d'entrée/sortie 6](#_Toc403547700)

[Fonctions de lecture/écriture d'un secteur 7](#_Toc403547701)

[Questions 7](#_Toc403547702)

[Bilan visite professeur 7](#_Toc403547703)

[Semaine 2 7](#_Toc403547704)

[Kernel en C, interruption et appels systèmes 7](#_Toc403547705)

[Infos 7](#_Toc403547706)

[Problèmes rencontré 7](#_Toc403547707)

[Questions 8](#_Toc403547708)

[Bilan visite professeur 8](#_Toc403547709)

[Etape 3 9](#_Toc403547710)

[Description détaillée 9](#_Toc403547711)

[Rôles 9](#_Toc403547712)

[Semaine 1 9](#_Toc403547713)

[Description structure système de fichier 9](#_Toc403547714)

[Implémentation structure du système 9](#_Toc403547715)

[Ajouter un fichier au système 9](#_Toc403547716)

[Lister les fichiers du système 10](#_Toc403547717)

[Supprimer un fichier du système 10](#_Toc403547718)

[Questions 10](#_Toc403547719)

[Bilan visite professeur 10](#_Toc403547720)

Création d’un mini OS simple en plusieurs étapes et par groupes.

# Etape 1

## Description détaillé

Implémenter un bootloader qui chargera un mini noyau à l’aide de l’émulateur QEMU.

* Créer une **image disque** qui servira de disque de boot à QEMU. Elle contiendra le secteur de boot suivi du mini kernel.
* Créer **2 fichiers en assembleur**, un pour le **bootloader** et un autre pour le **kernel**. Ils seront **compilés à l’aide d’un makefile**.
* Le **bootloader** devra **afficher** un message *« Loading kernel at address xxx... »* et **implémenter la lecture du kernel** une adresse choisie.
* Le **kernel** devra **afficher un message** permettant de vérifier qu’il a été correctement chargé.

## Rôles

* Chef de projet : Hottinger Jeremy
* Rédacteur : António Domingos Ana Sofia
* Présentateur : Argentin Yvann
* Expert : Lopes Marques Vasco

## Semaine 1

### Recherches :

Vu que nous ne savions pas comment appliquer la théorie vue en cours, nous avons fait quelques recherche et avons trouvé un tutoriel[[1]](#footnote-1) qui explique bien les choses et nous nous sommes basé sur lui pour commencer cette étape.

### Travail effectué :

* Chargement du bootloader en affichant un message avec l’interruption 0x10.
* Essai chargement du kernel avec l’interruption 0x13.
  + - Ne fonctionnait pas.

### Bilan visite professeur :

* Comprendre et arranger le problème du chargement du kernel.
* Bonne chose de tenir un cahier de bord.
* Lire comment faire un kernel en c.

## Semaine 2

### Travail effectué :

Problème chargement du kernel résolu.   
Le problème venait de l’ordre des mov dont il faut faire très attention

### Bilan visite professeur :

* Utiliser int 13/42h
* Initialiser la pile
* Faire attention sur quel secteur nous mettons nos données.
* Plus de rôles spécifiques
* Commencer l’étape 2
* Explication étape 2 (notes manuscrite Ana)

# Etape 2

## Description détaillée

Implémenter un kernel en C plutôt qu’en assembleur.

Développement du code de base permettant d’afficher du texte à l’écran, de lire des caractères frappés au clavier et de lire/écrire des secteurs sur le disque.

Puis mettre en place un mécanisme permettant de réaliser des appels systèmes.

Finalement, implémenter des appels systèmes de base.

## Rôles

* Modification bootloader et stack : Jérémy Hottinger
* Kernel en C, interruption et appels systèmes : Yvann Argentin
* Fonctions d'entrée/sortie : Vasco Lopes Marques
* Lecture/écriture d'un secteur: Ana Sofia Domingos

## **Semaine** 1

### Modification bootloader et stack

#### Description

Modifier l’interruption permettant de load le kernel en utilisant int 13/42h.

Initialisation de la stack car elle sera nécessaire pour le reste du projet.

### Kernel en C, interruption et appels systèmes

#### Description

En s’aidant du kernel en assembleur, implémenter le kernel en C.

Implémenter la fonction interrupt qui permet d’utiliser une interruption lors de l’appel des fonctions d’entrée/sortie et de lecture/écriture d’un secteur.

Implémenter la fonction init\_syscall pour affecter des appels systèmes.

### Fonctions d'entrée/sortie

#### Description

Implémentation de la fonction print\_string qui permet d’afficher une chaîne de caractères.

Implémentation de la fonction read\_string qui permet de lire les chaînes de caractères tapées par l’utilisateur.

### Fonctions de lecture/écriture d'un secteur

#### Description

Implémentation de la fonction read\_sector qui permet de lire un secteur dont l’utilisateur aura choisi le numéro.

Implémentation de la fonction write\_sector qui permet de vérifier la fonction read\_sector en y ajoutant un fichier texte (avec dd) puis de le lire et l’afficher avec la fonction print\_string.

#### Recherches

J’ai fait quelques rechercher et j’ai trouvé un site[[2]](#footnote-2) expliquant bien comment fonctionne les secteurs ainsi que comment implémenter la lecture de secteurs.

### Questions

* Est-ce que je dois appeler mon kernel.c via un kerner.asm ?
* Si non, comment charger le kernel.c ?
* Comment initialiser les segments ? Et utiliser l’interruption 42

### Bilan visite professeur

* Laisser la modification de l’interruption 42 pour plus tard.
* Implémenter les fonctions
* Faire marcher le kernel en C.

## **Semaine** 2

### Kernel en C, interruption et appels systèmes

Nous avons codé nos fonctions d’entrée/sorties ainsi que les fonctions de secteurs.

Mais nous avons quelques problèmes que nous n’arrivons pas à résoudre.

### Infos

.global : permet de voir en dehors de l’assembleur.

.extern : permet d’appeler les fonctions en externe dans le code c.

### Problèmes rencontré

* Problème avec le linker. (résolu)
* Init\_Syscall ne marche pas. (résolu)
* Si on met nos fonctions sur le fichier kernel.c, ça marche plus ou moins. (résolu)
* On n’arrive pas à lire l’entrée clavier. (résolu)
* On n’arrive pas à tester le bon fonctionnement des fonctions de secteur.

### Questions

* Comment tester le read\_sector avec un fichier ? et en utilisant le print\_string ?
* Le pointeur buffeur doit se remplir du contenu du secteur ? puis il faut le lire ?
* Comment faire marcher l’init\_syscall ?
* Quel doit être l’ordre les fichiers dans le linker ?

### Bilan visite professeur

* Explication étape 3 (notes manuscrites Ana)
* Laisser les fonctions de secteurs de côté pour la 3e partie pour l’instant
* Implémenter les fonctions linux

# Etape 3

## Description détaillée

Implémenter un système de fichier simple dans notre kernel.

Il permettera d’ajouter des fichiers, lister les fichier et supprimer des fichiers.

Le système de ficher à une structure spécifique. (cf cahier des charge étape 3)

## Rôles

Nous avons créé un projet sur taiga.io[[3]](#footnote-3) qui nous permet de créer des sprints et se répartir les tâches en fonction du travail à effectuer.

## Semaine 1

### Description structure système de fichier

|  |  |  |
| --- | --- | --- |
| **Structure** | **Nom** | **Type** |
| SuperBlock | Signature | char[8] |
| NbSecteur | int[2] |
| TailleBitmap (en block) | int[1] |
| TailleFileEntry | int[256] |
| BlockReservedFileEntry | int[16] |
| BlockReservedContent | int[16] |
| Bitmap | Tab de 1024 entrés | char[1024] |
| FileEntry | Name | char[32] |
| Size | int |
| TabIndexes | int[(256-34)/2] |
| FileContent |  |  |

### Implémentation structure du système

A vous de mettre à jour

### Ajouter un fichier au système

#### Infos

Le nom du fichier ne doit pas dépasser 32 caractères de long.

La taille est représentée sur 16 bits.

Le premier block des tabIndexes à l’index où se trouve le contenu dans le fileContent ou à 0 pour indiquer qu’il est vide.

Le contenu du fichier peut être fractionné sur plusieurs blocks du fileContent.

J’ai codé une fonction qui permet d’ajouter un fichier à la structure du système de fichiers et ajouter de file entries.

### Lister les fichiers du système

J’ai codé une fonction c qui permet de parcourir les structures du système de fichier et d’afficher les infos des FileEntries.

### Supprimer un fichier du système

A vous de mettre à jour

### Questions

* Est-ce que le nombre de FileEntries est fixe et certain sont vide ou est-ce qu’elles sont ajoutées au fur et à mesure ?

### Bilan visite professeur

1. <http://viralpatel.net/taj/tutorial/hello_world_bootloader.php> [↑](#footnote-ref-1)
2. <http://www.brokenthorn.com/Resources/OSDev5.html> [↑](#footnote-ref-2)
3. <https://tree.taiga.io/project/malecommex-osp3a/backlog> [↑](#footnote-ref-3)