António Domingos Ana Sofia

**Cahier de bord**

Argentin Yvann

Hottinger Jeremy

Lopes Marques Vasco

Table des matières

[Etape 1 3](#_Toc401088627)

[Description détaillé 3](#_Toc401088628)

[Rôles 3](#_Toc401088629)

[Semaine 1 3](#_Toc401088630)

[Recherches : 3](#_Toc401088631)

[Travail effectué : 3](#_Toc401088632)

[Codage load bootloader : 4](#_Toc401088633)

[Bilan visite professeur : 4](#_Toc401088634)

[Semaine 2 4](#_Toc401088635)

[Travail effectué : 4](#_Toc401088636)

[Codage load kernel : 5](#_Toc401088637)

[Codage kernel : 5](#_Toc401088638)

[Bilan visite professeur : 6](#_Toc401088639)

[Etape 2 6](#_Toc401088640)

[Description détaillée 6](#_Toc401088641)

[Rôles 6](#_Toc401088642)

[Semaine 1 6](#_Toc401088643)

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

# 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

* Problème avec le linker.
* Init\_Syscall ne marche pas.
* Si on met nos fonctions sur le fichier kernel.c, ça marche plus ou moins.

### 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

1. <http://viralpatel.net/taj/tutorial/hello_world_bootloader.php> [↑](#footnote-ref-1)
2. <http://www.brokenthorn.com/Resources/OSDev5.html> [↑](#footnote-ref-2)