António Domingos Ana Sofia

**Cahier de bord**

Argentin Yvann

Hottinger Jeremy

Lopes Marques Vasco

Table des matières

[Etape 1 4](#_Toc410160080)

[Description détaillé 4](#_Toc410160081)

[Rôles 4](#_Toc410160082)

[Semaine 1 4](#_Toc410160083)

[Recherches : 4](#_Toc410160084)

[Travail effectué : 4](#_Toc410160085)

[Bilan visite professeur : 4](#_Toc410160086)

[Semaine 2 5](#_Toc410160087)

[Travail effectué : 5](#_Toc410160088)

[Bilan visite professeur : 5](#_Toc410160089)

[Etape 2 6](#_Toc410160090)

[Description détaillée 6](#_Toc410160091)

[Rôles 6](#_Toc410160092)

[Semaine 1 6](#_Toc410160093)

[Modification bootloader et stack 6](#_Toc410160094)

[Kernel en C, interruption et appels systèmes 6](#_Toc410160095)

[Fonctions d'entrée/sortie 6](#_Toc410160096)

[Fonctions de lecture/écriture d'un secteur 7](#_Toc410160097)

[Questions 7](#_Toc410160098)

[Bilan visite professeur 7](#_Toc410160099)

[Semaine 2 7](#_Toc410160100)

[Kernel en C, interruption et appels systèmes 7](#_Toc410160101)

[Infos 7](#_Toc410160102)

[Problèmes rencontré 7](#_Toc410160103)

[Questions 8](#_Toc410160104)

[Bilan visite professeur 8](#_Toc410160105)

[Etape 3 9](#_Toc410160106)

[Description détaillée 9](#_Toc410160107)

[Rôles 9](#_Toc410160108)

[Semaine 1 9](#_Toc410160109)

[Description structure système de fichier 9](#_Toc410160110)

[Implémentation structure du système 9](#_Toc410160111)

[Ajouter un fichier au système 9](#_Toc410160112)

[Lister les fichiers du système 10](#_Toc410160113)

[Supprimer un fichier du système 10](#_Toc410160114)

[Questions 10](#_Toc410160115)

[Bilan visite professeur 10](#_Toc410160116)

[Semaine 2 10](#_Toc410160117)

[Fonctions de lecture/écriture d'un secteur 10](#_Toc410160118)

[Fonctions du système de fichier 10](#_Toc410160119)

[Bilan visite professeur 10](#_Toc410160120)

[Semaine 3 10](#_Toc410160121)

[Fonctions du système de fichier 10](#_Toc410160122)

[Semaine 4 et 5 11](#_Toc410160123)

[Infos 11](#_Toc410160124)

[Fonction get\_stat dans le kernel 11](#_Toc410160125)

[Fonction read\_file dans le kernel 11](#_Toc410160126)

[Fonction remove\_file dans le kernel 11](#_Toc410160127)

[Fonction iterator dans le kernel 11](#_Toc410160128)

[Semaine 6 11](#_Toc410160129)

[Mise en commun 11](#_Toc410160130)

[Optimisation 11](#_Toc410160131)

[Etape 4 12](#_Toc410160132)

[Semaine 1 12](#_Toc410160133)

[Implémentation shell 12](#_Toc410160134)

[Implémentation mini librairie C 12](#_Toc410160135)

[Semaine 2 à 4 12](#_Toc410160136)

[Implémentation fonction exec 12](#_Toc410160137)

[Implémentation shell 12](#_Toc410160138)

[Implémentation mini librairie C 12](#_Toc410160139)

[Mise en commun 12](#_Toc410160140)

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 fichiers et supprimer des fichiers.

Le système de ficher à une structure spécifique. (cf cahier des charges 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

Création d’un fichier header contenant la structure declarée ci-dessus.

Remplissage des valeurs par défaut du système de fichier. (sfscreate)

### 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

Nous modifions le bitmap correspondant en le mettant à 0. Et nous mettons un 0 au début du nom du fichier pour indiquer qu’il est supprimé.

Nous ne touchons pas au fileContent.

### 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

* Faire complètement marcher l’étape 2
* Restructurer nos dossiers
* Commenter nos codes
* Prendre son bootloader

## Semaine 2

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

Nous avons réussi à faire fonctionner la lecture et l’écriture d’un ou plusieurs secteurs avec l’interruption 42.

Nous avions un problème avec la lecture, nous passions l’adresse du buffeur dans la structure, donc il ne savait pas quand s’arrêter de lire. En enlevant le &, nous sommes capable de nous arrêter de lire et de effectuer d’autres interruptions après.

### Fonctions du système de fichier

Pour le bitmap, nous n’avions pas remplit les bits de chaque bitmap donc nous avions modifié ceci.

Ce qui entraine d’adapter tous nos codes.

### Bilan visite professeur

* Délai de l’étape 3 repoussé d’une semaine
* Faire fonctionner l’étape 3

## Semaine 3

### Fonctions du système de fichier

Adaptation de nos fonctions pour utiliser les bits du bitmap au lieu d’un bitmap.

Mise en commun de nos fonctions.

Création de l’image contenant le système de fichiers.

## Semaine 4 et 5

### Infos

* Un block vaut 1024, lors de la lecture dans le kernel il faudra lire 2 secteurs (512) pour lire un block entier.
* Une fileEntry vaut 256 donc dans chaque secteur dans la partie fileEntry nous avons 2 fichiers. (fileEntry)

### Fonction get\_stat dans le kernel

Récupère les informations liées à un fichier spécifié. Cette fonction utilise la fonction iterator.

### Fonction read\_file dans le kernel

Lit le contenu du fichier spécifié. Pour cela il faut :

* Récupérer les fileContent utilisé en fonction des tabIndexes du fileEntry.
* Aller dans le bon fileContent et récupérer le contenu du fichier
* Remplir un buffer avec le contenu complet du fichier dans le système de fichier

### Fonction remove\_file dans le kernel

Efface un fichier spécifié. Pour cela, il faut :

* Récupérer les bitmap utilisé en fonction des tabIndexes du fileEntry.
* Mettre le premier bit du nom du fichier à 0
* Aller dans le bitmap et mettre à 0 le bon bit correspondant

### Fonction iterator dans le kernel

Itère sur tous les fileEntry

## Semaine 6

### Mise en commun

* Nous avons mis nos 4 fonctions dans le même fichier
* Nous avons effectué des tests pour vérifier le bon fonctionnement des fonctions.

### Optimisation

* Création d’un makefile par repertoire
* Des fonctions read et write sector. C’est devenu une seule fonction à laquelle nous passons le numéro de l’interruption (42 ou 43)
* Création d’une nomenclature contenant tous les defines dont nous avons besoin
* Ajout de commentaires uniforme sur tous les fichiers.
* Implémentation d’une fonction modulo et division entière en assembleur

# Etape 4

## Semaine 1

### Implémentation shell

Lit les caractères entrés en ligne de commande. En fonction des commandes entrées, il faut :

* Récupérer les paramètres s’il y en a
* Appeler les fonctions correspondantes

### Implémentation mini librairie C

Appel des fonctions que nous avons implémenté dans le kernel pour que l’utilisateur puisse les appeler et les utiliser dans un programme.

## Semaine 2 à 4

### Implémentation fonction exec

Nous avons créé un code utilisateur que nous avons compilé en bin.  
Nous lisons le contenu de ce fichier pour le mettre en adresse mémoire en utilisant la fonction memputb. Pour cela nous parcourons le contenu ce fichier et nous mettons byte par byte en changeant d’offset dans le seg passée en paramètre à la fonction execute.

Après avoir mis tout le fichier en mémoire nous allons jump.

Malheureusement, nous n’arrivons pas à exécuté le code lors du jump car lors de la lecture du fichier nous ne lisons pas le fichier en entier car il y a un \0 dans le fichier et la lecture ne va pas plus loin. Pourtant dans le code de lecture de fichier nous devons lire tout un secteur et pas s’arrêter aux \0.

### Implémentation shell

En fonction de la commande tapée en dans le terminal, nous allons interpréter cette commande et appeler la fonction correspondante.

### Implémentation mini librairie C

Création d’un header contenant les defines dont nous aurons besoin et les structures.

Codage des fonctions qui vont travailler avec les strings.

Lien de la libc aux interruptions.

### Mise en commun

Link la libc avec le shell et le code utilisateur. Nous n’avons pas pu tester car exec ne fonctionne pas.

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)