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.

### Codage load bootloader :

|  |
| --- |
| *;------------- Initialize segments -------------;*  *MOV SI, data ;Store string pointer to SI*  *CALL printString ;Call print string procedure*  *JMP kernelLoader ;Infinite loop, hang it here.*  *;------------- display message function -------------;*  *printCharacter: ; Procedure to print character on screen*  *MOV AH, 0x0E ; Set registers to display a message*  *MOV BH, 0x00*  *MOV BL, 0x07*  *INT 0x10 ; Call video interrupt*  *RET*  *printString: ; Procedure to print string on screen*  *next\_character:*  *MOV AL, [SI] ; Store byte in AL*  *INC SI ; Increment SI*  *OR AL, AL ; Check if end of string*  *JZ exit\_function ; if end, return*  *CALL printCharacter ; else print char*  *JMP next\_character ; Get next char*  *exit\_function:*  *RET* |

### 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

### Codage load kernel :

|  |
| --- |
| *kernelLoader:*  *; Set registers to read sectors from a disk*  *mov ah, 0x02 ; Read Disk Sectors*  *mov al, 0x01 ; Nb sectors to read*  *mov ch, 0x00 ; Track*  *mov cl, 0x02 ; Sector*  *mov dh, 0x00 ; Head*  *mov dl,dl ;*  *mov bx, 0x1000 ; Set Segment to 0x1000 (futur kernel address)*  *mov es, bx ;*  *mov bx, 0x0000 ; Offset value*  *readsector:*  *int 0x13 ; BIOS interruption*  *jc readsector ; if error retry*  *mov ax, es ; Set the data segment register*  *mov ds, ax ; to point to the kernel location in memory*  *jmp 0x1000:0x0000 ; Jump to the kernel*  *;Data*  *data db 'Loading kernel at adress 0x1000', 10, 13, 0*  *; fill with nop instructions until offset 510*  *times 510-($-$$) db 0x90*  *; last 2 bytes of sector: indicate a bootable sector*  *dw 0xAA55* |

### Codage kernel :

|  |
| --- |
| kernel:  ;----------définition représentation à l'écran---------;  MOV AH, 0x0E ; mode teletype  MOV BH, 0x00 ; mode page zero  MOV BL, 0x07 ; gris léger  MOV SI, HelloString ; place l'adresse de la phrase dans   le registre SI  Call PrintString ; appel la fonction d'impression    ;-------------- Fonction Impression --------------------;  PrintString: ;  next\_character: ;  MOV AL, [SI] ; Met le contenu du pointeur SI dans registre AL  OR AL, AL ; Compare si zéro  JZ exit\_function ; Si oui on quitte  Call PrintCharacter ; Si non, appel impression du   charactère à l'écran  INC SI ; On passe à l'adresse suivante  jmp next\_character ; On revient au début de la boucle  exit\_function:  RET  PrintCharacter: ; impression à l'écran  INT 0x10  RET  ;------------------ Bloc de données -------------------;  HelloString db 'Loading KERNEL', 0; |

### Bilan visite professeur :

# Etape 2

## Description détaillée

Créer un kernel en C.

## Rôles

* Chef de projet :
* Rédacteur :
* Présentateur :
* Expert :

## **Semaine** 1

1. <http://viralpatel.net/taj/tutorial/hello_world_bootloader.php> [↑](#footnote-ref-1)