Mini-OS x86

Architecture des Systèmes d'Exploitation

Quentin Pointeau

Sciences du Numérique
Architecture, Systèmes et Réseaux
ENSEEIHT
29 mai 2025

1 Introduction

Ce rapport vise à recenser les fonctionnalités apportées à la base du code fourni pour ce mini système d'exploitation x86 ainsi que les choix de conception et d'implémentation pour certaines des fonctionnalités mentionnées ci-dessous.
Il est à noter que j'ai nommé mon mini système d'exploitation QOS, en référence à mon prénom (QuentinOS) ainsi qu'à la Quality of Service, notion essentielle dans l'étude des réseaux.

2 Console

La première fonctionnalité apporté à la base de code du projet fut une console, donc un affichage pour notre système d'exploitation.
Le choix a été fait de ne retenir qu'un index pour la position du curseur, à partir de laquelle on peut calculer la ligne et la colonne sur lesquelles se trouve actuellement le curseur. Plusieurs tests ont été effectués avec moult appels à `printf`. D'autres améliorations ont été apportées à la console mais étant donné qu'elles sont liées à l'apport de fonctionnalités développées plus tard, nous les verrons dans leurs catégories respectives.

3 Pagination

La pagination, c'est LA fonctionnalité de ce projet.
Le choix retenu pour l'implantation de la gestion de la mémoire physique a été d'utiliser une bitmap afin de réduire la taille du stockage de l'état des pages physiques étant donnée que l'information *"une page est-elle libre ou non ?"* est binaire. Une fonction `print_mem` a été implantée afin de visualiser les disponibilités des pages et ainsi faire du debug.
Rien de particulier pour l'implantation des pages en mémoire virtuelle mais voici les tests effectués :

```
alloc_page_entry(0xA000FFFC, 1, 1);
uint32_t *ptr = (uint32_t*)0xA000FFFC;
uint32_t test = *ptr;
test++;
```

Ce test conduit à une segmentation fault car on se place à la fin d'une page mémoire virtuelle et on essaie d'écrire sur une nouvelle page qui n'a pas été allouée.

4 Interruptions

Rien de particulier dans l'implantation des interruptions. L'ajout d'un handler pour l'interruption 50 a été implanté et voici le test effectué :

```
__asm__ ("int $50"); // Permet de déclencher l'interruption 50
Affichage : Incredible! I've caught the interruption number 50!
```

5 Timer

La mise en place d'un timer est maintenant possible grâce à l'implantation des interruptions. En effet, le timer que j'ai implanté n'est autre qu'un compteur qui s'incrémente à chaque interruption timer (n°32). Afin d'afficher le timer à l'écran, j'ai décidé de modifier la console afin de créer une sorte de *"taskbar"* qui est affichée en permanence sur la première ligne de l'écran. Une de mes améloirations personnelles a été de synchroniser le timer avec l'heure réelle du système en allant chercher le jour, la date et l'heure depuis le BIOS. Plus précisément, dans `time.c`, je récupère les informations du CMOS grâce aux ports `0x70` (commande) et `0x71` (données) puis le timer continue à fonctionner sans aucune autre synchronisation nécessaire : la synchronisation ne se fait qu'une seule fois à l'initialisation du timer. Ainsi, ces nouvelles fonctionnalités ont conduit à des ajouts de fonctions utilitaires dans `console.c` afin d'afficher la date et l'heure sur la taskbar.
`https://wiki.osdev.org/CMOS`

6 Appels systèmes

Rien de particulier dans l'implantation des appels systèmes. J'ai bien remplacé l'appel de `console_putbytes` de `printf` par l'appel à `sys_write`. Les tests effectués sont donc :

```
// Shutdown the system
shutdown(1);

printf("\fWelcome to QOS !\n\n");
```

7 Processus

Après les appels systèmes, il a fallu implanter les processus.
Plusieurs choix de conception ont été faits. J'ai tout d'abord décidé de stocker les processus dans un tableau de longueur `NB_PROC` (défini à 255). Comme pour les pages mémoires physique, j'ai choisi d'utiliser une bitmap afin de répertorier les processus existants. Les PID des processus se créent de manière incrémentale, la seule limite étant la capacité d'un entier non signé sur 32 bits soit plus de 4 milliards de PID de processus disponibles. Le dernier PID utilisé est stocké dans la variable `current_pid`.

On peut créer, bloquer et débloquer un processus. Cependant, le blocage des processus est permanent jusqu'à qu'on le débloque, il n'y a pas de notion de temps de blocage. Lors de l'initialisation des processus dans `init_proc`, le processus 0 `idle` est créé et est mis à l'état `ELECTED` donc c'est celui qui tourne en premier sur le système. Ce processus `idle` crée ensuite deux processus : `proc1` et `minishell`.

L'algorithme de scheduling implanté est un tourniquet qui réparti également le temps d'exécution des processus (Round Robin). A chaque interruption timer, la variable `current_quantum` est incrémenté modulo `QUANTUM`. Et à chaque fois que `current_quantum` passe à 0, la fonction `schedule` est appelée. Ici j'ai décidé de mettre `QUANTUM` à 1 ce qui fait que `schedule` est appelée toutes les 100 millisecondes (le timer étant en millisecondes, j'appelle `update_quantum` dans le handler du timer quand `timer % 10` vaut 0).

Pour tester les processus, j'ai utilisé `printf` dans chacun des trois processus pour voir à l'affichage le déroulement du tourniquet.

Pour ce qui est des améliorations personnelles, j'ai implanté (après avoir implanté le fonctionnement du clavier que l'on verra juste après) un minishell qui permet simplement à l'utilisateur de pouvoir effacer l'écran, afficher la liste des processus en cours d'exécution, de bloquer un processus, de débloquer un processus, de tuer un processus (malheureusement non fonctionnel), d'éteindre le système et de quitter le `minishell` (ce qui revient à être bloqué car `idle` et `proc1` ne font rien). Les touches du clavier sont stockées dans un buffer jusqu'à ce que l'utilisateur appuie sur la touche `ENTER` auquel cas la commande est comparée avec celle définis puis exécutée le cas échant.

8 Clavier

La dernière étape de projet a été de faire en sorte que l'utilisateur puisse utiliser le clavier (fonctionnalité que j'ai développé juste avant de créer le minishell).

Un buffer est nécessaire pour stocker les codes de touches entrées par l'utilisateur et fournis par le handler de l'interruption du clavier (interruption 33). Pour ce buffer, j'ai choisi comme préconisé de le faire de manière circulaire. Ainsi, j'ai un index qui indique le début du buffer et un autre index qui indique la fin du buffer.

Pour le clavier, ma seule amélioration personnelle a été de prendre en compte la touche `RSHIFT` car ne pas pouvoir l'utiliser était frustrant. Ainsi, j'ai deux variables booléennes dans le handler de l'interruption clavier qui stockent respectivement l'état d'appui de la touche `LSHIFT` et l'état d'appui de la touche `RSHIFT`.

9 Conclusion

Pour conclure, quand on regarde d'où on est parti, grâce à la base de code du projet, on se rend compte du chemin parcouru. L'implantation de la console, de la pagination, des interruptions, du timer, des appels systèmes, des processus et du clavier auront été des bons exercices de réflexion sur la conception d'un système d'exploitation du point de vue kernel.