Rapport de projet OS 2019 - Rock'n Roll

Maximilien Dupont de Dinechin et Eloi Massoulié

1. Objectifs

L'objectif était d'implémenter un OS minimal, appelé `OSselet`, sous forme d'un microkernel et d'un shell. Nous n'avons pas atteint tous nos objectifs, loin de là, mais voici un descriptif de ce que l'on souhaitait réaliser.

Pour le kernel : - démarrer sur une architecture virtuelle `x86` dans `qemu`, - être indépendant des fonctions C non standard en les réecrivant dans `lib/`, de sorte à pouvoir cross-compiler, - être capable d'afficher des choses sur l'écran en sortie VGA (notamment en réecrivant `printf`), - être capable de gérer une entrée clavier standard avec une gestion minimaliste des interruptions, - fournir un système de fichier minimal : un seul dossier, création et suppression de fichiers, écriture, concarénation...

Pour le shell : - Quelques commandes simples (echo, ls, pwd, touch, ...), - Exécution de scripts shell basiques (pas fait)

2. Réferences

En plus du cours, deux sources principales : OSDev et Kernel 101 (et 201) sur https://arjunsreedharan.org.

3. Compilation

La compilation du projet suppose d'avoir un cross-compiler GCC, dont l'installation est documentée ici : https://wiki.osdev.org/GCC_Cross-Compiler.

Le fichier Makefile décrit la compilation automatisée du projet avec `make`, il n'y a plus qu'à lancer `qemu` avec l'option `-kernel OSselet.bin`.

4. Structure du projet

Le dossier `kernel/` contient le minimum vital : le code assembleur permettant de charger la fonction kernel_main, le code c correspondant, et le linker.

Le dossier `lib/` contient une réimplémentation de quelques fonctions des librairies standard C ; en plus de la gestion écran et clavier (`tty/`)

Le dossier `usr/` contient les applications utilisateurs : une calculatrice (inechevée), et le shell (dans `usr/shell/`).

Le dossier `mem/` contient le système de fichiers (et un embryon de malloc).

4.1 `kernel/`

4.1.1 Démarrage, chargement du kernel

Le fichier `kernel.c` réalise les initialisations de bibliothèques élémentaires (pour le shell, les interruptions et le système de fichiers décrits ci-après), puis se met en attente d'un signal de l'utilisateur. Nous avons de plus laissé la possibilité de rentrer des instructions pour le shell dans la fonction elle-même, sans passer par l'interface utilisateur.

4.1.2 Affichage

L'affichage VGA se fait en écivant dans le tableau à l'adresse `0xB8000`. Les fonctions d'affichage permettent de simplifier le processus d'écruture de caractères à l'écran : retour à la ligne, décalage de lignes vers le bas pour faire de la place...

En pratique, ces opérations ne sont plus utilisées directement, mais automatiquement lors des appels à `printf`.

4.1.3 Entrée clavier

Une gestion basique d'interruption permet d'attendra une saisie clavier, en activant l'écoute des ports `0x20` et `0x21`. Le basculement dans le mode saisie au clavier fonctionne, avec la prise en charge du backspace et l'arrêt de l'input à la saisie du retour chariot ; mais la sortie de l'interrupt reste problématique, et nous n'arrivons pas à retourner au point d'interruption. Pour cette raison, une seule entrée clavier par démarrage de l'OS fonctionne.

(pour tester : écrire `echo test` puis retour, la commande est bien interprétée et exécutée, mais le contrôle n'est pas rendu à la fonction kernel_main.)

Pour cette raison, les appels aux fonctions shell, (pour démontrer par exemple le fonctionnement du fs), se font directement dans le code de `kernel.c`, à l'aide de la fonction `kernel_eval`.

4.2 `lib/`

Ce dossier contient les réimplémentations de quelques fonctions des bibliothèques standard `string.h`, `stdio.h`, `math.h`. Les fonctions sont chacune contenue dans leur propre fichier C, et les headers regroupés dans `lib/include/`.

4.3 `usr/shell/`

Le shell proposé est très basique. Il se découpe en un lexer minimaliste, qui lit les lignes en attendant une commande en premier mot, et des arguments dans les mots suivants. À terme, nous aurions voulu avoir une gestion de l'écriture avec `>>>`, ou du chaînage de commandes, donc l'implémentaion est préparée mais non encore fonctionnelle.

4.4 `mem/`

Notre système de fichiers ne reposait pas sur un modèle préexistant : les fichiers sont contenus dans une unique chaîne de caractères `files`, tandis qu'un second tableau `sommaire` contient les descriptions de chaque fichier (rangs de début et de fin dans `files`, booléen indiquant si chaque emplacement est libre). Avec ce modèle, nous avons développé quelques fonctions rudimentaires de création, lecture et manipulation de fichiers :

- `create` prend en entrée une chaîne de caractères et un nom à lui donner, et la crée dans `files`. Cela implique notamment de trouver un emplacement libre assez grand, puis de répertorier ce nouveau fichier dans `sommaire`. L'adresse dans le sommaire du fichier créé est renvoyée ;
- `lookup` est un auxiliaire permettant de trouver un fichier dans le sommaire en fonction de son nom ;
- `remove` prend un mot et marque son emplacement comme libre ;
- `append` est une concaténation grossière : il supprime le premier mot et en crée un nouveau constitué des deux mots en entrée ;
- `cat` renvoie la chaîne de caractères correspondant au fichier demandé ;
- `ls` parcourt `sommaire` et affiche les noms associés à tous les emplacements libres ;
- `cleanup` est une fonction de nettoyage de la mémoire, qui utilise les deux précédentes `moveword` et `shift`. Lorsqu'elle est appelée, les emplacements libres adjacents sont fusionnés et déplacés vers la fin.

En l'état, les tests sont rendus difficiles par des dysfonctionnements dans `create`, développés plus loin.

5. Conclusions et problèmes rencontrés

Il semble que beaucoup de problèmes dans l'exécution du shell viennent du manque d'un vrai malloc : l'allocation automatique de C pour les string semble une création de nombreux conflits. En effet, l'affichage de messages de débug en cours d'éxecution était suffisant pour modifier en cours de route les paramètres d'appels de fonctions. C'est un problème dont nous n'avions pas soupçonné l'existence, et dont la découverte trop tardive a été un handicap pour mener à bien la communication entre le shell et le système de fichiers.

Quelques incompréhensions autour de l'implémentation concrète des interruptions nous ont également empêché de mettre en place la communication que nous aurions aimé avoir avec le clavier : support des majuscules, ou bien même tout simplement rendre le contrôle au programme interrompu.

De manière générale, l'aspect "concret" des choses a été une source de nombreuses difficultés : même en ayant écrit un malloc ou un système de fichiers fonctionnels

dans la ram, nous n'avons pas réussi à les rattacher au "hardware".