retour au descriptif des séances "système"?

Boot et premier programme en mode kernel

Préambule

Principes pédagogiques

Le but des TD est de préparer le travail que vous devez faire dans le TP. L'idée générale des TP est de créer, très progressivement, un tout petit système d'exploitation. Évidemment, dans le temps imparti, il n'est pas envisageable de créer un système complexe. Ce système est petit, mais il se veut simple à comprendre.

Pour présenter les concepts des systèmes d'exploitation, la méthode employée est en général *top-down*. On vous présente les services des systèmes (gestion des fichiers, gestion des processus, gestion des communications interprocessus, etc.), puis on vous présente comment un système open source tel que Linux fait pour rendre ces services. C'est très intéressant, mais le système pris comme base est tellement complexe, qu'il est nécessaire de ne voir qu'une petite partie, et certains étudiants perdent la vue d'ensemble. Pour l'UE d'architecture des ordinateurs, c'est une approche impossible parce qu'elle est trop éloignée de l'architecture matérielle.

Si Linux est trop complexe alors pourquoi ne pas prendre un petit système ad hoc, mais en conservant l'approche *top-down* ? Oui, cela peut être envisagé, c'est d'ailleurs ce qui a été fait dans un ancien module. Toutefois, C'est encore difficile, parce que pour bien comprendre comment fonctionne un service du système d'exploitation, il faut

avoir une vue d'ensemble du système et ce n'est simple à présenter.

Nous avons choisi, une approche *bottom-up*. Nous partons de rien, et nous ajoutons progressivement les services en limitant le nombre de fichiers et la taille des codes. Chaque nouveau service qui s'ajoute s'appuie sur les services précédemment construits.

Pour conclure cette présentation des principes, si on devait vous apprendre comment est faite une voiture. L'approche *top-down* consiste à prendre une voiture et à la démonter pour voir de quoi elle est faite. L'approche *bottom-up* consiste à assembler des éléments pour construire une toute petite voiture (genre 2CV :-)).

Principe des séances

1. Chaine de compilation, boot du système, pilote de périphériques, bibliothèque de fonctions pour le noyau.
2. Lancement d'une application utilisateur, bibliothèque système, gestionnaire de syscall.
3. Parallélisme de tâches sur un seul processeur en temps partagé.

Pour le rendre simple à comprendre, nous introduisons progressivement les éléments en plusieurs étapes. Pour cette première séance, il y a 5 étapes décrites ci-après. Avant de le décrire.

1. Un petit programme de quelques lignes en assembleur, placé entièrement dans la région mémoire du boot, qui réalise l'affichage du message "Hello World". C'est un tout tout petit programme, mais pour obtenir l'exécutable, vous devrez utiliser tous les outils de la chaîne de cross-compilation MIPS et pour l'exécuter vous devrez exécuter le simulateur du prototype. C'est simple, mais c'est nouveau pour beaucoup d'entre vous.
2.

1. Premier programme en assembleur dans la seule section de boot

Codes

```
    .section .boot,"ax",@progbits       // def. of a new section: .boot (see https://bit.ly/3gzKWob)
                                        // flags "ax":     a -> allocated means section put in memor
                                        //                 x -> section contains instructions
                                        // type @progbits: contains somethings (not just space)
boot:
    la      $26,    kinit               // get address of kinit() function
    j       $26                         // goto kinit()
```

tp2_hcpu.s

Objectif

- Affichage d'un message sur le terminal avec un programme en assembleur.

Compétences acquises

- Savoir produire un exécutable à partir d'un code en assembleur.

- Savoir comment afficher un caractère sur un terminal.

Questions

- Dans quel fichier se trouve la description de l'espace d'adressage du MIPS ?
- Dans quel fichier se trouve le code de boot et pourquoi l'avoir nommé ainsi ?
- A quelle adresse démarre le MIPS ?
- Que produit le compilateur C quand on utilise l'option -c ?
- Que fait l'éditeur de liens ?
- De quels fichiers a besoin l'éditeur de liens pour fonctionner ?
- Dans quelle section se trouve le code de boot pour le compilateur ?
- Dans quelle section se trouve le message hello pour le compilateur ?
- Dans quelle section se trouve le code de boot dans le code exécutable ?
- Dans quelle région de la mémoire le code de boot est placé ?
- Comment connaît-on l'adresse du registre de sortie du contrôleur de terminal TTY ?
- Comment sait-on que le message est fini et que le programme doit s'arrêter ?
- Pourquoi terminer le programme par un `dead: j dead` ?

2. Saut dans le code du noyau en assembleur

Objectif de l'étape

- Affichage d'un message depuis le code du noyau toujours en assembleur

Compétences acquises

- Comment aller à une adresse définie dans un autre fichier
- Création d'une section dans le code objet produit par le compilateur

Questions

- Quel est le nom de la directive assembleur permettant de déclarer une section

3. Saut dans la fonction kinit() du noyau en langage C

Objectif de l'étape

-

Compétences acquises

-

Questions

-

4. Accès aux registres de contrôle des terminaux TTY

Objectif de l'étape

-

Compétences acquises

-

Questions

-

3. Fonction d'initialisation en C

Objectif de l'étape

-

Compétences acquises

-

Questions

-