Architecture des ordinateurs 1 - L3

Ce site est dédié aux 3 dernières séances des modules Architecture des ordinateurs (LU3NI029 et LU3NI129) de la licence d'informatique de Sorbonne Université.
Vous y trouverez :

1. les documents sur le MIPS : mode user et mode kernel et celui sur l'environnement des TP
2. les objectifs généraux des séances en lien avec ce que vous avez déjà vu
3. le fonctionnement des séances de TD-TP, c'est-à-dire ce qui vous est demandé
4. La présentation du contenu des séances.

A. Documents sur le MIPS et sur l'environnement de TP

Vous devez lire ou au moins parcourir ces 3 documents : **A1** (MIPS/user), **A2** (MIPS/kernel) et **A3** (environnement TP).

Vous connaissez déjà le document **A1**, mais pour pouvoir bien comprendre les notions vues lors des 3 séances, vous devez lire les deux autres. Par exemple, le MIPS fonctionne dans deux modes : le mode user et le mode kernel. Le mode kernel permet aux programmes d'accéder à toutes les ressources de l'ordinateur, ce que cela signifie en détail est présenté dans le document **A2**. Autre exemple, vous devez exécuter vos programmes sur un prototype virtuel d'ordinateur. La configuration de votre environnement de travail est dans le document **A3**.

Pour vous aider, le plan des sections de ces documents est recopié ci-après.

- **A1. Documentation MIPS32 architecture et assembleur en mode user ?**
    1. Registres de l'architecture externe accessible en mode user (p. 2)
    2. Espace d'adressage du MIPS32 (p. 4)
    3. Syntaxe et principales directives du langage assembleur (p. 5)
    4. Codage des instructions utilisateur du MIPS32 (p. 9)
    5. Instructions accessibles en mode utilisateur (p. 11)
    6. Appels système de simulateur de processeur **Mars** (p. 22)
    7. Convention d'appel des fonctions (p. 24)

- **A2. Documentation MIPS32 architecture et assembleur en mode kernel ?**
    1. Modes d'exécution du processeur MIPS **?**

2. Registres protégés utilisables seulement en mode kernel **?**
3. Découpage de l'espace d'adressage **?**
4. Instructions protégées **?**
5. Cause d'entrée et de sortie du noyau du système d'exploitation **?**
6. Fonctionnement du registre d'état `c0_sr` **?**
7. Fonctionnement du registre de cause `c0_cause` **?**

- **A3. Configuration de l'environnement des TP ?**
  1. Environnement de travail pour les TP
  2. Prototype virtuel et chaîne de compilation MIPS
  3. Compilation et exécution des programmes

B. Objectifs généraux des 3 séances

Les **premières séances de l'UE** décrivent l'architecture externe du MIPS (celle visible du programmeur) et la programmation structurée en assembleur (avec des fonctions et une pile). Les programmes réalisés utilisent des structures de données simples telles que des tableaux à une dimension et des enregistrements (les *struct* du C). L'accès aux entrées-sorties se fait par des demandes de services en utilisant l'instruction `syscall` du MIPS. Les programmes sont exécutés sur le simulateur de processeur **MARS** permettant d'observer l'évolution des registres du processeur et l'évolution des segments de mémoire utilisés par le code, les data et la pile.

Dans cette première partie, l'architecture de l'ordinateur n'est pas détaillée et ce que fait l'instruction `syscall` n'est pas détaillé non plus. Le but des 3 dernières séances est d'étudier plus en détail l'architecture d'un ordinateur simple, de type ?microcontrôleur à base de MIPS, et d'y exécuter une application au-dessus d'un embryon de système d'exploitation qui exécute les appels système, c'est-à-dire `syscall`.

Concernant le matériel, il est composé d'un processeur MIPS connecté à une mémoire et quelques périphériques. La mémoire contient le code et les données. Les périphériques sont les composants permettant les entrées-sorties (p. ex. le terminal écran-clavier) ou alors les composants offrant un service spécifique (p. ex. le *timer* qui compte le temps). Vous allez :

1. Manipuler les deux modes d'exécution du processeur (mode *kernel* et mode *user*) ;
2. Communiquer avec les contrôleurs de périphériques grâce à leurs registres de commandes en utilisant les instructions *load*/*store* (`lw`/`sw`);
3. Utiliser les requêtes d'interruption des contrôleurs de périphériques pour interrompre le programme en cours d'exécution.

Concernant le logiciel, il est écrit principalement en langage C et un peu en langage assembleur. Il est composé d'un empilement de couches logicielles. Il y a tout d'abord le code de démarrage du processeur (le *boot*), puis le noyau du système d'exploitation, puis la bibliothèque système (la *libc*), et enfin l'application de l'utilisateur. Vous allez :

1. Programmer en langage C pour le noyau en utilisant la chaîne de compilation standard (compilateur + éditeur de liens) via un `Makefile` ;
2. Exécuter les programmes sur un simulateur d'ordinateur complet avec processeur, mémoire et contrôleurs de périphériques ;
3. comprendre quelques services du noyau tels que les gestionnaires de syscall et d'interruptions.

Pour présenter les concepts des systèmes d'exploitation, nous avons choisi, une approche *bottom-up*. Nous partons d?une feuille blanche, 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.

Il n'y a que trois séances, c'est donc juste le début. La suite est vue au second semestre dans l'UE Archi 2 (LU3NI031) pour ceux que cela intéresse.

C. Fonctionnement des séances de TD et TP

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. Ce système est petit, mais il se veut simple à comprendre.

Toutes les séances sont structurées de la même manière. Chaque séance est découpée en étapes qui doivent être suivies dans l'ordre. Chaque étape est indépendante des autres du point de vue des fichiers, c'est-à-dire qu'elle n'utilise pas les fichiers des étapes précédentes et donc s'il y a des fichiers en commun entre les étapes, ceux-ci sont répliqués. Le code fourni est toujours fonctionnel et il y a toujours un `Makefile` pour produire l'exécutable et le faire tourner sur le simulateur du prototype d'ordinateur. Le code est très commenté et il n'y a pas ? ou peu ? de "trous" à remplir.

Chaque étape introduit un petit nombre de concepts. Dans la première étape de la première séance, il n'y a que 2 fichiers sources (`hcpu.S` et `kernel.ld`) et 1 `Makefile` simple (de type *collection de scripts Shell*). Dans les étapes suivantes, on ajoute progressivement des fichiers avec les services de l'OS et on complexifie le Makefile.

Le travail demandé pour chaque étape est le suivant :

1. Il faut répondre à des questions de la partie TP portant sur le code et sur l'architecture. Les réponses aux questions sont en général dans le code ou dans les commentaires du code ou dans les réponses aux questions du TD qui a préparé la séance de TP ou enfin, dans les diapositives du cours.
   Le but de ces questions est de pointer les difficultés introduites dans l'étape et de les associer à des éléments de réponses vus précédents. Par exemple, dans la première étape, une question est : « - *Quelle est l'adresse en mémoire de la première instruction du MIPS?* ». La réponse est dans le cours et dans le TD et dans le fichier `ldscript.ld`. La réponse est « *L'adresse de la première instruction est* `0xBFC00000` ».

2. Il faut ajouter une fonctionnalité au programme dans un ou plusieurs fichiers et décrire cet ajout dans le compte-rendu. Cet ajout peut nécessiter la création d'un nouveau fichier source.
   Le but de cet ajout est de s'approprier le code grâce à une petite modification et d'en voir toutes les implications. Par exemple, dans la première étape de la première séance, il est demandé d'afficher deux messages au lieu d'un.

D. Séances

? Démarrage de l'ordinateur

Cours 9

- Architecture d'un SoC minimal (SoC = ordinateur intégré sur une puce), celui que vous allez utiliser en TP.
- Chaîne de compilation du langage C puisque vous allez utiliser le langage C
- Présentation des piles de couches logicielles pour comprendre comment l'application communique avec le système d'exploitation
- Présentation du prototype virtuel du SoC que vous allez utiliser en TP

TD 9 et TP 9

- L'architecture de l'ordinateur utilisé contient un MIPS, une mémoire et d'un contrôleur de terminaux.
- Comment un ordinateur simple démarre.
- Comment le programmeur peut interagir avec le monde extérieur via les contrôleurs de périphériques.
- Comment utiliser *GCC* avec un `Makefile` pour produire le noyau.

? Application en mode user

Cours 10

- Modes d'exécution du MIPS
- Composants du noyau et de la libc
- Communication entre kernel.x et user.x
- Visite guidée du code sur un exemple

TD 10 et TP 10

- Les deux modes d'exécution du processeur, le mode *kernel* et le mode *user*.
- Comment produire deux exécutables: `kernel.x` pour le noyau et `user.x` pour l'application.
- Comment se passe le passage du noyau à l'application et inversement.
- Comment fonctionne le gestionnaire d'appel système.

? Gestion des interruptions

Cours 11

- Définition des interruptions
- Vue matérielle des interruptions
- Vue logicielle des interruptions

TD 11 et TP 11

- Ajout d'un composant *timer* qui active périodiquement une ligne d'interruption
- Ajout d'un concentrateur de lignes d'interruption qui rassemble toutes les lignes d'interruptions.
- Comment fonctionne le gestionnaire des interruptions.