Cette page décrit la séance complète : TD et TP. Elle commence par des exercices à faire sur papier et puis elle continue et se termine par des questions sur le code et quelques exercices de codage simples à écrire et à tester sur le prototype. La partie pratique est découpée en 4 étapes. Pour chaque étape, nous donnons (1) une brève description, (2) une liste des objectifs principaux de l'étape, (3) une liste des fichiers avec un bref commentaire sur chaque fichier, (4) une liste de questions simples dont les réponses sont dans le code, le cours ou le TD et enfin (5) un exercice de codage.

IMPORTANT
Avant de faire cette séance, vous devez avoir lu les documents suivants :

- Séance de TME sur le démarrage du prototype? : *obligatoire*
- Cours sur l'exécution d'une application en mode user : *obligatoire*
- Document sur l'assembleur du MIPS et la convention d'appel des fonctions : *recommandé*
- Documentation sur le mode kernel du MIPS32 : *obligatoire*

Récupération du code du TP

- Téléchargez **l'archive code du tp2** et placez là dans le répertoire `$HOME/AS5`
- Assurez-vous que vous avez déjà sourcé le fichier `Source-me.sh` (sinon lisez ?Configuration de l'environnement des TP ? Étape 3)
- Ouvrez un `terminal`, allez dans le répertoire `AS5` (**`cd ~/AS5`**) et décompressez l'archive du tp1 avec **`tar xvzf tp2.tgz`**
  Cette étape est peut-être inutile si vous avez déjà fait la décompression de l'archive au moment de son téléchargement.
- Dans le `terminal`, exécutez la commande **`cd ; tree -L 2 AS5`**. Vous devriez obtenir ceci (tp1 et tp2):

```
AS5
??? bin
?   ??? almo1.x
?   ??? gcc
?   ??? Source-me.sh
?   ??? test
?   ??? tracelog
??? tp1
?   ??? 1_hello_boot
?   ??? 2_init_asm
?   ??? 3_init_c
?   ??? 4_nttys
?   ??? 5_driver
```

```
?   ??? Makefile
??? tp2
    ??? 1_klibc
    ??? 2_appk
    ??? 3_syscalls
    ??? 4_libc
    ??? Makefile
```

Objectif de la séance

Cette séance illustre le cours2. Les applications de l'utilisateur s'exécutent en mode user. Dans la séance précédente, nous avons vu que les registres de commande des contrôleurs de périphériques sont placés dans l'espace d'adressage du processeur. Les adresses de ces registres ont été placées dans la partie de l'espace d'adressage interdite en mode user. Ainsi, une application n'a pas un accès direct aux périphériques, elle doit utiliser des appels système (avec l'instruction syscall) pour demander au noyau du système d'exploitation. C'est ce que nous allons voir. Le code est désormais découpé en 4 couches logicielles :

1. `1_klibc` le code de boot (utilisé seulement au démarrage);
2. `2_appk` le noyau du système d'exploitation, ici pour l'essentiel, la fonction d'initialisation `kinit()` et le gestionnaire des appels systèmes;
3. `3_syscalls` la bibliothèque de fonctions standards (libc);
4. `4_libc` l'application.

A. Travaux dirigés

A1. Les modes d'exécution du MIPS

Dans cette section, nous allons nous intéresser à ce que propose le processeur MIPS concernant les modes d'exécution. Ce sont des questions portant sur l'usage des modes en général et le comportement du MIPS vis-à-vis de ces modes. Dans la section A3, nous verrons le code de gestion des changements de mode; Le MIPS propose deux modes d'exécution.

Questions

1. Le MIPS propose deux modes d'exécution, rappelez quels sont ces deux modes et à quoi ils servent? (*Nous l'avons dit dans le descriptif de la séance*).
2. Commencez par rappeler ce qu'est l'espace d'adressage du MIPS, puis dîtes ce que veut dire qu'une adresse X mappée en mémoire, et enfin dîtes si une adresse X mappée en mémoire est toujours accessible (en lecture ou en écriture) quelque soit le mode d'exécution du MIPS.
3. Le MIPS propose des registres à usage général (GPR *General Purpose Register*) pour les calculs ($0 à $31). Le MIPS propose un deuxième banc de registres à l'usage du système d'exploitation, ce sont les registres système. Comment sont-ils numérotés? Chaque registre porte un nom correspondant à son usage, quels sont ceux que vous connaissez, donner leur nom, leur numéro et leur rôle? Peut-on faire des calculs avec des registres? Quelles sont les instructions qui permettent de les manipuler?
4. Le registre status est composé de plusieurs champs de bits qui ont chacun une fonction spécifique. Décrivez le contenu du registre status et le rôle des bits de l'octet 0(seulement ceux vu en cours).
5. Le registre cause est contient la cause d'appel du kernel. Dites à quel endroit est stockée cette cause et donnez la signification des codes 0, 4 et 8
6. Le registre `EPC` est un registre 32 bits qui contient une adresse. Vous devriez l'avoir décrit dans la question 2, mais expliquez pourquoi ce doit être l'adresse de l'instruction qui provoque une exception qui doit être

stocké dans `EPC`?
7. Nous avons vu trois instructions ne sont pas utilisable lorsque le MIPS est en mode kernel, lesquelles? Que font-elles? Est-ce que `syscall` peut-être utilisée en mode user?
8. Quelle est l'adresse d'entrée dans le noyau?
9. Que se passe-t-il quand le MIPS entre dans le noyau, après l'exécution de l'instruction `syscall`?
10. Quelle instruction utilise-t-on pour sortir du noyau et entrer dans l'application ? Dîtes précisément ce que fait cette instruction dans le MIPS?

A2. Langage C pour la programmation système

La programmation en C, vous connaissez, mais quand on programme pour le noyau, c'est un peu différent. Il y a des éléments de syntaxe ou des besoins spécifiques.

Questions

1. définition de section pour le placement des fonctions ou des variables?
2. effacement des variables globales?
3. accès aux registres de périphériques?
4. ldscript?
5. inclusion de code assembleur en C?
6. appel de code assembleur depuis le code C?

A3. Passage entre les modes kernel et user

Le noyau et l'application sont deux exécutables compilés indépendamment mais pas qui ne sont pas indépendants. Vous savez déjà que l'application appelle les services du noyau avec l'instruction syscall, mais comment ça se passe vraiment depuis le code C et comment le noyau gère-t-il cet appel? En outre, il y a l'autre sens, comment le noyau lance-t-il l'application?

Questions

1. Comment imposer le placement d'adresse d'une fonction ou d'une variable en mémoire?
2. Convention utilisée pour que le noyau puisse lancer l'application?
3. Que faire avant et après l'exécution de la fonction `main()` ?
4. Quels sont les évènements traités par le noyau?
5. le registre se réserve l'usage de deux registres, pourquoi et lesquels?
6. Comment le noyau traite les causes d'invocation?
7. comment fonctionne le gestionnaire de syscall?
8. comment fonctionne le gestionnaire d'appel système?
9. quels sont les fichiers communs?

A4. Génération du code exécutable

Pour simuler le logiciel, il faut produire deux exécutables. Nous utilisons, ici, un Makefile hiérarchique et des règles explicites. Cela sort du cadre de l'architecture, mais vous avez besoin de ce savoir-faire pour comprendre le code, alors allons-y.

Questions

1. Rappelez là quoi sert un Makefile?
2. Comment appeler un makefile depuis une autre Makefile?
3. comment décrire une règle explicite?
4. comment utiliser les variables automatiques du C?

A5. Libc

Cette partie ne concerne pas vraiment le noyau, mais il y a peut-être des choses que vous ignorez sur le C, ou certaines opérations, qu'il est nécessaire de connaître pour ce petit système. Cela n'a pas été présenté en cours, alors les questions sont précédées d'une présentation du problème et sa solution.

Questions

1. fonction C à nombre d'arguments variables `fprintf`?
2. génération de nombres pseudo-aléatoire `rand`?
3. traduction d'une chaîne de caractère en nombre `atoi`?

B. Travaux pratiques

Pour les travaux pratiques, vous devez d'abord répondre aux questions, elles ont pour but de vous faire lire le code et revoir les points du cours. Les réponses sont dans le cours ou dans les fichiers sources. Certaines ont déjà été traitées en TD, c'est normal. Ensuite, vous passez aux exercices pratiques.

Le code se trouve dans `$AS5/tp2/`, ouvrez un terminal et allez-y. Dans ce répertoire, vous avez 4 sous-répertoires et un Makefile. Le fichier `$AS5/tp2/Makefile` permet de faire le ménage en appelant les Makefiles des sous-répertoires avec la cible `clean`.

B1. Ajout d'une bibliothèque de fonctions standard pour le kernel (klic)

Le noyau gère les ressources matérielles et logicielles utilisées par les applications. Il a besoin de fonctions standards pour réaliser des opérations de base, telles qu'une fonction `print` ou une fonction `rand`. Ces fonctions ne sont pas très originales, mais elles recèlent des subtilités que vous ne connaissez peut-être pas encore. En outre, nous allons utiliser un Makefile définissant un graphe de dépendance explicite entre les fichiers cibles et les fichiers source avec des règles de construction.

Objectifs

- Ajouter une bibliothèque de fonctions standards
- Utiliser un Makefile avec des règles explicites

Fichiers

```
1_klibc/
??? kinit.c         : fichier contenant la fonction de démarrage du noyau
??? harch.h         : API du code dépendant de l'architecture
??? harch.c         : code dépendant de l'architecture du SoC
??? hcpu.h          : prototype de la fonction clock()
```

```
??? hcpu.S          : code dépendant du cpu matériel en assembleur
??? kernel.ld       : ldscript décrivant l'espace d'adressage pour l'éditeur de lien
??? klibc.h         : API de la klibc
??? klibc.c         : fonctions standards utilisées par les modules du noyau
??? Makefile        : description des actions possibles sur le code : compilation, exécution, ne
```

Questions

1. En ouvrant tous les fichiers dessiner le graphe de dépendance de `kernel.x` vis-à-vis de ses sources?
2. ?

Exercices

- Ajout de la fonction cpuid() qui lit le registre $15 du coprocesseur système.

B2. Programme utilisateur mais exécuté en mode kernel

Nous allons désormais avoir deux exécutables: le noyau et l'application. Dans cette étape, nous allons voir comment le noyau fait pour appeler l'application, alors que celle-ci n'est pas compilée en même temps que le noyau.

Objectifs

Fichiers

```
2_appk/
??? Makefile        : Makefile racine qui invoque les Makefiles des sous-répertoires et qui exéc
??? kernel ?????????? Répertoire des fichiers composant le kernel
?   ??? kinit.c     : fichier contenant la fonction de démarrage du noyau
?   ??? harch.h     : API du code dépendant de l'architecture
?   ??? harch.c     : code dépendant de l'architecture du SoC
?   ??? hcpu.h      : prototype de la fonction clock()
?   ??? hcpu.S      : code dépendant du cpu matériel en assembleur
?   ??? klibc.h     : API de la klibc
?   ??? klibc.c     : fonctions standards utilisées par les modules du noyau
?   ??? kernel.ld   : ldscript décrivant l'espace d'adressage pour l'édition de liens du kernel
?   ??? Makefile    : description des actions possibles sur le code kernel : compilation et nett
??? user ??????????? Répertoire des fichiers composant l'application user
    ??? crt0.c      : fonctions d'interface entre kernel et user, pour le momment : crt0()
    ??? main.c      : fonction principale de l'application
    ??? user.ld     : ldscript décrivant l'espace d'adressage pour l'édition de liens du user
    ??? Makefile    : description des actions possibles sur le code user : compilation et nettoy
```

Questions

1. Question ?

B3. Programme utilisateur utilisé en mode user mais sans libc

Le programme utilisateur doit absolument s'exécuter en mode user et il doit passer par des appels système pour accéder aux services du noyau. Les services, ici, sont limités (l'accès au TTY, exit et clock), il n'empêche que pour gérer ces appels, il faut l'analyseur des causes d'appels à l'entrée du noyau et un gestionnaire de `syscall`. Il faut aussi le gestionnaire d'exceptions, parce que s'il y a une erreur de programmation, le noyau doit afficher quelque chose pour aider le programmeur.

Objectifs

Fichiers

```
3_syscalls/
??? Makefile        : Makefile racine qui invoque les Makefiles des sous-répertoires et qui exéc
??? common ?????????? répetoire des fichiers commun kernel / user
?   ??? syscalls.h  : API la fonction syscall et des codes de syscalls
??? kernel ?????????? Répertoire des fichiers composant le kernel
?   ??? kinit.c     : fichier contenant la fonction de démarrage du noyau
?   ??? harch.h     : API du code dépendant de l'architecture
?   ??? harch.c     : code dépendant de l'architecture du SoC
?   ??? hcpu.h      : prototype de la fonction clock()
?   ??? hcpu.S      : code dépendant du cpu matériel en assembleur
?   ??? klibc.h     : API de la klibc
?   ??? klibc.c     : fonctions standards utilisées par les modules du noyau
?   ??? kpanic.h    : déclaration du tableau de dump des registres en cas d'exception
?   ??? kpanic.c    : fonction d'affichage des registres avant l'arrêt du programme
?   ??? ksyscalls.c : Vecteurs des syscalls
?   ??? kernel.ld   : ldscript décrivant l'espace d'adressage pour l'édition de liens du kernel
?   ??? Makefile    : description des actions possibles sur le code kernel : compilation et nett
??? user ???????????? Répertoire des fichiers composant l'application user
    ??? crt0.c      : fonctions d'interface entre kernel et user, pour le momment : crt0()
    ??? main.c      : fonction principale de l'application
    ??? user.ld     : ldscript décrivant l'espace d'adressage pour l'édition de liens du user
    ??? Makefile    : description des actions possibles sur le code user : compilation et nettoy
```

Questions

1. Question ?

B4. Accès aux registres de contrôle des terminaux `TTY`

Objectifs

Fichiers

```
4_libc/
??? Makefile        : Makefile racine qui invoque les Makefiles des sous-répertoires et qui exéc
??? common ?????????? répetoire des fichiers commun kernel / user
?   ??? syscalls.h  : API la fonction syscall et des codes de syscalls
??? kernel ?????????? Répertoire des fichiers composant le kernel
?   ??? kinit.c     : fichier contenant la fonction de démarrage du noyau
?   ??? harch.h     : API du code dépendant de l'architecture
?   ??? harch.c     : code dépendant de l'architecture du SoC
?   ??? hcpu.h      : prototype de la fonction clock()
?   ??? hcpu.S      : code dépendant du cpu matériel en assembleur
?   ??? klibc.h     : API de la klibc
?   ??? klibc.c     : fonctions standards utilisées par les modules du noyau
?   ??? kpanic.h    : déclaration du tableau de dump des registres en cas d'exception
?   ??? kpanic.c    : fonction d'affichage des registres avant l'arrêt du programme
?   ??? ksyscalls.c : Vecteurs des syscalls
?   ??? kernel.ld   : ldscript décrivant l'espace d'adressage pour l'édition de liens du kernel
?   ??? Makefile    : description des actions possibles sur le code kernel : compilation et nett
??? uapp ???????????? Répertoire des fichiers de l'application user seule
?   ??? main.c      : fonction principale de l'application
?   ??? Makefile    : description des actions possibles sur le code user : compilation et nettoy
```

```
??? ulib ??????????? Répertoire des fichiers des bibliothèques système liés avec l'application
    ??? crt0.c      : fonctions d'interface entre kernel et user, pour le momment : crt0()
    ??? libc.h      : API pseudo-POSIX de la bibliothèque C
    ??? libc.c      : code source de la libc
    ??? main.c      : fonction principale de l'application
    ??? user.ld     : ldscript décrivant l'espace d'adressage pour l'édition de liens du user
    ??? Makefile    : description des actions possibles sur le code user : compilation et nettoy
```

Questions

1. Question ?