Architecture des **S**ystèmes d'**E**xploitation
Maîtrise d'Informatique

Janvier 2014

Devoir Surveillé

– tous documents autorisés –

Le langage C est requis pour les implémentations

Pensez à détailler vos raisonnements et vos calculs

Les différentes parties peuvent être traitées indépendamment, tous les documents sont autorisés, mais ne perdez pas trop de temps à chercher l'information utile. Notez encore que l'énoncé contient 24 points, je m'arrêterai à 20/20 ;-)

Exercice I. mémoire virtuelle (10 points)

Nous considérons ici le mécanisme de mémoire virtuelle fourni par le coprocesseur CP15 adjoint à un cœur de processeur ARM. Selon les spécifications de ce coprocesseur, l'adresse (virtuelle) 32 bits sollicitée par le microprocesseur est traduite en adresse physique selon la configuration de la MasterTranslationTable. Pour cela les 12 bits de poids forts, appelés I1Offset, sont utilisés comme indexe de lecture de cette table. Les entrées de la MasterTranslationTable sont de la forme suivante :

```
struct T1Entry {
  union {
    struct {
      unsigned base : 12 ;
      unsigned RFU : 18 ;
    } singleStepWalking ;
    struct {
      unsigned secondLevelPage : 20 ;
      unsigned RFU : 10 ;
    } twoStepWalking ;
  } ;
  unsigned entryType : 2 ;
} ;
```

Pour mémoire, les champs d'une struct sont les uns à la suite des autres, en mémoire, alors que les champs d'une union sont les uns « sur » des autres :
`struct S{ int a ; int b } var1;`
`union U{ int a; int b } var2;`

adresse	valeur
&var1	var1.a
&var1+4	var1.b

adresse	valeur
&var2	var2.a, var2.b
&var2+4	-

Sur cet exemple, en modifiant `var2.a` vous modifiez aussi `var2.b` …

En fait ces entrées peuvent être configurées de deux usages différents :
Lorsque `entryType == 1` c'est la sous-structure `struct singleStepWalking` de l'union anonyme qui doit être exploitée. Dans ce cas l'adresse physique est formée en remplaçant les 12 bits de poids fort de l'adresse virtuelle (précédemment nommés I1Offset) par les 12 bits définis par le champ `base` de la sous-structure `struct singleStepWalking` de l'entrée I1Offset de la table `MasterTranslationTable`. Les 22 bits de poids faible de l'adresse virtuelle restent inchangés dans l'adresse physique produite.
Lorsque `entryType == 2` c'est la sous-structure `struct twoStepWalking` de l'union anonyme qui doit être exploitée. Dans ce cas les bits 21 à 10 de l'adresse virtuelle, nommés `I2Offset`, sont utilisés comme indexe de lecture dans une table de second niveau. Cette table de second niveau dont l'adresse de base est donnée par le champ `secondLevelPage`, suivi de 12 zéro pour former une adresse sur 32 bits. A l'entrée I2Offset de cette table se trouve la structure suivante :

```
struct T2Entry {
  unsigned base : 22 ;
  unsigned NYS  : 10 ;  } ;
```

L'adresse physique finalement accédée est formée des 22 bits de poids fort du champ `base` *de la structure* `T2Entry` *suivis des 10 bits de poids faible de l'adresse virtuelle.*

Question I.1 (1 point)

Selon la spécification du coprocesseur CP15, combien d'entrées du type `struct T1Entry` la table `MasterTranslationTable` contient-elle ?

Question I.2 (1 point)

Selon la déclaration de la structure `struct T1Entry`, quelle est la taille, en octet, d'une entrée de la table ? (justifiez votre réponse)

Question I.3 (1 point)

En conséquence, quelle est la taille totale, toujours en octets, de la table `MasterTranslationTable` ?

Question I.4 (2 points)

Lorsque `entryType` est égal à 2, le fonctionnement de la MMU proposée par le coprocesseur CP15 est assez comparable au fonctionnement de la MMU d'un Intel tel qu'il a été présenté en cours. Mais lorsque `entryType` est égal à 1 la MMU a un fonctionnement assez sensiblement différent. Selon vous pourquoi lorsque `entryType` est égal à 1 l'accès à la mémoire virtuelle est globalement plus rapide que lorsque ce champ est égal à 2 ?

Question I.5 (1 point)

Lorsque `entryType` est égal à 1, quelle est la taille d'une page de mémoire ?

Question I.6 (1 point)

Lorsque `entryType` est égal à 2, quelle est la taille d'une page de mémoire ?

Question I.7 (1 points)

Quelle est la taille d'une table de niveau 2, tel qu'elle est pointée par une entrée de la `MasterTranslationTable` avec un champ `entryType` égal à 2 ?

Question I.8 (2 points)

Si toutes les entrées de la table `MasterTranslationTable` ont un champ `entryType` égal à 2 et pointent vers différentes tables d'indirection de second niveau, quelle est la taille totale de la mémoire occupée par des tables destinées à traduire des adresses virtuelles en adresses physique ? (expliquez votre calcul).

Exercice II. Ordonnancement de type « Rate Monotonic » (8 points)

L'algorithme d'ordonnancement Rate monotonic (i.e. « à taux monotone ») est un ordonnanceur utilisé dans les systèmes dit temps réel. Selon une version simplifiée de cet algorithme, chaque tâche définit une charge utile, que se « résume » à un pourcentage d'occupation du microprocesseur. L'ordonnancement consiste à donner à chaque tâche un intervalle de temps proportionnel à la charge utile. Dans le contexte de cet exercice, la charge utile qui est une valeur entre 1 et 100 correspond au temps, en milliseconde que l'ordonnanceur donne à l'exécution d'une tâche. Lorsque le temps consacré à la tâche est dépassé, l'ordonnanceur élit la tâche suivante, pour une durée égale à sa charge (toujours en milliseconde). Ainsi, si une tâche réclame 2% elle sera sure d'avoir au moins 2 millisecondes toutes les 100 millisecondes.
Pour mémoire, voici ci-dessous un exemple fichier prototype de l'ordonnanceur (qui utilisait un algorithme d'ordonnancement « Round Robin ») réalisé en TP :

```
1.  #define CTX_MAGIC 0xCAD0AB0B
2.
3.  typedef void (t_fct) (void *);
4.
5.  enum ctx_state_e { /* enum des etats d'un contexte     */
6.    CTX_READY,       /*  . Contexte activable           */
7.    CTX_ACTIVABLE,   /*  . Contexte re-activable        */
8.    CTX_TERMINATED   /*  . Contexte termine             */
9.  };
10.
11. struct s_ctx {
12.   unsigned int     magic ;    /* detrompeur            */
13.   void            *savedESP ; /* valeur courante d'ESP */
14.   void            *savedEBP ; /* valeur courante d'EBP */
15.   t_fct           *startfct ; /* fonction de depart    */
16.   void            *arg ;      /* argument de depart    */
17.   char            *stack ;    /* pile d'execution      */
18.   enum ctx_state_e state ;    /* etat courant          */
19.   struct s_ctx    *nextCtx ;  /* chaînage pour "ring"  */
20. } ;
21.
22. /* anneau des contextes ordonnancables                */
23. struct ctx_s * ring      = NULL ;
24. /* contexte d'exécution actuellement actif            */
25. struct s_ctx *currentCtx = NULL ;
26.
27. /* Creation d'un nouveau contexte à ordonnancer       */
28. int initCtx(struct s_ctx *aCtx, int stackSize, t_fct f,
       void *arg) ;
29. /* Implementation de la strategie d'ordonnancement    */
30. void yield() ;
31. /* Implem. du mecanisme de commutation de contexte    */
32. void switchToCtx(struct s_ctx *aCtx) ;
33. /* demarrage de la préemption de tâche                */
34. void setupCtx() ;
```

listing Ctx.h

```
1.  [...]
2.  void yield() { /* implémentation du "round-robin" */
3.    switchToCtx(ring->next);
4.  }
5.  [...]
```

Implémentation de yield() vue en TP.

Question II.1 (2 point)

La première difficulté, dans l'implémentation d'un ordonnanceur à taux monotone tel qu'il a été spécifié, est de s'assurer de « d'ordonnançabilité» de l'ensemble des tâches qui ont été créées. Pour qu'un ensemble de tâches temps réel soit ordonnançable il faut et suffit que l'ordonnanceur puisse garantir que chacune aura (au moins) le pourcentage du temps d'exécution qu'elle a réclamé. Donnez la condition d'ordonnançabilité de n tâches associées à n charges notés C_i avec $i \in [1,n]$.

Question II.2 (2 point)

Pour que les tâches puissent fonctionner selon l'algorithme à taux monotone, le programmeur doit pouvoir les créer en précisant une charge associée à la tâche, puis l'ordonnancement doit pouvoir tenir compte de cette charge. Proposez un nouveau fichier de prototype qui tient compte de ces éléments pour assurer un ordonnanceur utilisant l'algorithme à taux monotone. Vous pourrez simplement indiquer ce qui change dans le fichier « `ctx.h` » donné précédemment.

Question II.3 (2 point)

Dans la Question II.1, vous avez définit une condition d'ordonnançabilité, qui lorsqu'elle n'est pas validée rend impossible l'ordonnancement d'un ensemble de tâches temps-réel par l'ordonnanceur à taux monotone. Proposez une évolution de l'implémentation de la fonction qui permet de créer des tâches de telle sorte que cette fonction refuse la réaction d'une nouvelle tâche lorsqu'elle ne permet plus de respecter la condition d'ordonnançabilité définie précédemment.

Question II.4 (2 points)

Dans la version que vous avez implémenté en TP, la fonction `setupCtx()` démarre l'ordonnancement préemptif en programmant une horloge matérielle qui déclenchera une interruption, appelant la fonction `yield()`. Cependant dans cette version de l'ordonnanceur, le temps d'exécution d'une tâche avant la prochaine préemption n'est pas fixé une fois pour toute (à 1 milliseconde par exemple), mais défini par la tâche. Avec l'algorithme à taux monotone, l'ordonnanceur ne change de tâche que lorsque le temps associé à la tâche (sa charge) est atteint.
Donner une implémentation de la stratégie d'ordonnancement qui correspond à cet algorithme à taux monotone (au minimum, vous devez donc redéfinir `setupCtx` et `yield)`.

Exercice III. Gestion asynchrone des accès au disque (6 points)

Nous nous intéressons ici à la gestion asynchrone d'un disque. Les registres d'accès au contrôleur de disque sont décrits dans l'extrait ci-dessous :
Extrait du fichier `Hardware.ini` qui décrit les ports associés au disque dur maître.

```
#
# Configuration des disques durs
#

# > Disque dur IDE Maitre
ENABLE_HDA      = 1     # ENABLE_HD=1 => simulation du disque activée
HDA_CMDREG      = 0x3F6 # registre de commande du disque maître
HDA_DATAREGS    = 0x110 # base des registres de données
                        # (r,r+1,r+2,...r+7)
HDA_IRQ         = 14    # Interruption du disque

# > Disque dur IDE Esclave
ENABLE_HDB      = 1     # ENABLE_HD=1 => simulation du disque activée
HDA_CMDREG      = 0x3F8 # registre de commande du disque maître
HDA_DATAREGS    = 0x112 # base des registres de données
                        # (r,r+1,r+2,...r+7)
HDA_IRQ         = 13    # Interruption du disque
```

L'envoi de commandes se fait également en écrivant sur le port désigné comme port de commande (maître ou esclave) dans le fichier de configuration. Cela permet au microprocesseur de solliciter une opération du disque magnétique choisi. Une fois que le microprocesseur envoie une commande, l'exécution de celle-ci débute immédiatement. Si la commande nécessite des données, il faut obligatoirement les avoir fournies (via les registres de données) avant de déclencher la commande. De la liste des commandes ATA-2 nous avons retenu le sous-ensemble décrit dans la table 1.

Nom	Code	Port de données (P0; P1; ...; P15)	objet
SEEK	0x02	numCyl (int16); numSec (int16)	déplace la tête de lecture
READ	0x04	nbSec (int16)	Lit nbSec secteurs
WRITE	0x06	nbSec (int16)	écrit nbSec secteurs
FORMAT	0x08	nbSec (int16); val (int32)	initialise nbSec secteurs avec val
STATUS	0x12	IRQFlags (int8)	Drapeau d'activation des interruptions.

Table 1 : sous-ensemble de commandes ATA-2

Dans cet exercice il s'agit d'exploiter les deux disques simultanément pour obtenir une fonction de backup (sauvegarde) du premier disque, sur le deuxième, aussi rapide que possible. Dans cet exercice nous considérons que les deux disques ont la même géométrie. Les disques considérés ont 512 secteurs de 64Ko et 8192 pistes. Le disque tourne à 7812,5 tours par seconde. La transition d'une piste à la suivante, ou à la précédente, prend 30 microsecondes.

Question III.1 (1 point)

Selon les spécifications données, quelle est la capacité de stockage des disques (en Ko) ?

Question III.2 (1 point)

Combien de temps faut-il pour pouvoir parcourir l'ensemble des secteurs de l'ensemble des pistes d'un disque ?

Question III.3 (2 points)

En pratique, une difficulté supplémentaire tient au fait que lorsqu'on programme un changement de piste, le disque continu à tourner. Il n'est donc généralement pas possible de passer du dernier secteur d'une piste n au premier secteur d'une piste n+1 sans faire un tour de disque supplémentaire. Car le temps que la nouvelle piste soit atteinte, le premier secteur est dépassé. Calculez sur quel secteur se trouve la tête de lecture lorsqu'on programme un changement de piste, après avoir atteint le dernier secteur de la piste précédente. Quelle conséquence cela a sur le programme qui parcourt l'ensemble d'un disque ?

Question III.4 (2 points)

Ecrivez une fonction qui copie entièrement le disque maître sur le disque esclave, le plus efficacement possible. Pour cela, assurez-vous d'une part, d'exploiter le parcours optimal des disques, que vous avez proposé dans la question précédente, et d'autre part, assurez-vous que les opérations de lecture sur le disque maître soient exécutées en « parallèle » avec les opérations d'écritures sur le disque esclave.