Architecture des **S**ystèmes d'**E**xploitation
Maîtrise d'Informatique et GMI-3

Janvier 2009

Devoir Surveillé

– tous documents autorisés –

Le langage C est requis pour les implémentations

Les différences 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'arrêterais à 20/20 ;-)

Partie I. Virtualisation du microprocesseur (4 points)

Nous nous intéressons ici à la mise en œuvre d'un ordonnanceur de tâches. Pour cela le matériel propose un support pour l'horloge. Les registres d'accès à l'horloge interne sont décrits dans l'extrait du fichier `Hardware.ini` ci-dessous :

```
#
# Configuration de l'horloge interne
#
TIMER_CLOCK = 0xF0 # registre de lecture de la date courante
TIMER_ALARM = 0xF4 # registre TIMER
TIMER_PARAM = 0xF8 # registre de configuration du TIMER
                   # bit 4 : RESET general (=1)
                   # bit 3 : Alarm ON = 1, Alarm OFF = 0
                   # bit 2 : Déclenche la division Hz du Timer (=1)
                   # bit 1 \ Si la division Hz du timer est demandé :
                   # bit 0 / 00: 1 top alarme pour 2 top d'horloge,
                   #         01: 1 top alarme pour 16 tops d'horloge,
                   #         10: 1 top alarme pour 128 tops d'horloge,
                   #         11: 1 top alarme pour 512 tops d'horloge.
TIMER_IRQ   = 2    # Niveau d'interruption de l'horloge
```

La fonction `int _in(int port);` réalise la lecture sur le port désigné. La valeur retournée correspond à la valeur qui a été lue sur ce port matériel. Les numéros de port sont identifiés dans le fichier de configuration du matériel `Hardware.ini`. La fonction `void _out(int port, int value);` réalise l'écriture d'une valeur sur le port désigné.
De plus, `void _sleep(int irqlvl);` suspend l'activité du microprocesseur jusqu'à l'occurrence d'une interruption de niveau au moins égal à `irqlvl`. Par ailleurs `IRQVECTOR` est un tableau de pointeur de fonction (de la forme `void func();`). Lorsqu'une interruption de niveau `n` est déclenchée par le matériel, c'est la fonction `IRQVECTOR[n]();` qui est exécutée, puis le programme en cours reprend son activité normale. Enfin `void _mask(int irqlvl);` permet d'inhiber l'appel à la fonction `IRQVECTOR[irqlvl]()` sur l'occurrence d'une interruption de niveau inférieure ou égale a `irqlvl`.
L'horloge interne est incrémentée 4 770 000 fois par seconde (cadencée à 4,77MHz). Il est possible de positionner la valeur (16 bits) du registre TIMER. Cette valeur sera

périodiquement incrémentée si le bit 3 du registre de configuration du TIMER_PARAM est positionné à 1. Par défaut le registre TIMER_ALARM est incrémenté en même temps que le registre TIMER_CLOCK. Cependant cette incrémentation peut avoir lieu tout les 2, 16, 128 ou même 512 tops d'horloges. Le « diviseur » d'incrémentation est fixé par les bits 0 et 1 du registre TIMER_PARAM à condition que le bit 2 soit positionné. Si l'incrémentation du registre TIMER_ALARM produit comme résultat la valeur 0xFFFF alors l'horloge interne génère une interruption (niveau 2). A l'incrément suivant le registre TIMER_ALARM vaut 0x0000 puis 0x0001…

Question I.1 – 1 point

Proposez fonction `void init_timer()` qui initialise l'horloge interne de manière à produire une interruption au bout de 20 millisecondes. Expliquez le calcul qui vous a amené à configurer les registres de l'horloge interne tel que vous les avez initialisés. Enregistrez la fonction `void irq_timer()` de tel manière qu'elle soit appelée au bout de 20 millisecondes.

Question I.2 – 1 point

Expliquez pourquoi la configuration proposée par la fonction que vous avez proposé en Question I.1, `void init_timer()`, ne suffit pas pour que `irq_timer()` soit appelée **toutes les** 20 millisecondes. Donnez une implémentation de `irq_timer()` qui assure que cette fonction soit automatiquement rappelée toutes les 20 millisecondes.

Question I.3 – 1 point

Nous supposons maintenant que nous disposons de la fonction C d'ordonnancement `void yield()` dont l'implémentation à été vue en TD/TP. Cette fonction ne délivre un résultat correct que si elle n'est pas interrompue par elle-même. Proposez une solution pour assurer qu'aucune interruption n'aura lieu pendant l'exécution de `yield()`.

Question I.4 – 1 point

Donnez une nouvelle implémentation de `void irq_timer()` de tel sorte que toutes les 20 millisecondes l'ordonnanceur change la tâche active.

Partie II. Fiabilité du partage du microprocesseur (12 points)

Le code suivant fournit un mécanisme d'exclusion mutuelle.

```
struct ctx_s { ...
  struct mutex_s *mutexList; /* mutex owned by the ctx             */
  struct ctx_s *lockList;    /* next task locked on the same mutex */
} ;

struct ctx_s currentCtx;     /* the current task                   */
...

struct mutex_s {
 int locked;                 /* TRUE when the mutex is locked      */
 struct ctx_s *owner;        /* the owner ctx of this mutex        */
 struct ctx_s *lockList;     /* List of tasks locked on the mutex  */
 struct mutex_s *mutexList;  /* next mutex owned by the same ctx   */
} ;

void create_mutex(struct mutex_s *mutex){       /* create a mutex */
  mutex->locked = 0;           /* the mutex is unlock             */
  mutex->owner = NULL;         /* nothing own it                  */
  mutex->lockList = NULL;      /* nothing is locked by mutex      */
  mutex->mutexList = NULL; /* mutex is not in a ctx requested List */
}

void mutex_request(struct mutex_s *mutex) {       /* lock a mutex */
  if(mutex->locked) {          /* if the mutex in not available   */
    currentCtx->state = Blocked;/* current task is suspended       */
                    /* add the current task in the mutex lockList */
    currentCtx->lockList = mutex->lockList;
    mutex->LockList = currentCtx;
    yield();                   /* schedule another task           */
  } else {                     /* else, if the mutex is available */
    mutex->owner = currentCtx;  /* currentCtx is the mutex owner   */
                /* the mutex is added to the mutexList of the ctx */
    mutex->MutexList = currentCtx->mutexList;
    currentCtx->mutexList=mutex;
    mutex->locked = 1;         /* the mutex is no more available  */
  }
}

void mutex_release(struct mutex_s *mutex) {      /* unlock a mutex */
  if(mutex->locked) {          /* if the mutex is not available   */
          /* remove the mutex form the mutexList requested by ctx */
    removeFromMutexList(currentCtx, mutex);
    mutex->owner = NULL;
    if(mutex->lockList) {      /* and (at less) a task is locked  */
      if(!mutex->lockList)  /* if nothing is waiting for the mutex */
        mutex->locked = 0;     /* the mutex is now unlocked       */
      else {                   /* else if something is pending    */
        mutex->lockList=Ready;  /* we can unlock a locked task    */
        mutex->owner=mutex->lockList;/* it is the new mutex owner. */
                  /* the next task is now the first waiting task */
        mutex->lockList=mutex->lockList->lockList;
      }
    }
  }
}
```

Question II.1 – 1 point

En considérant un mécanisme d'ordonnancement implicite (la procédure `yield()` qui active une nouvelle tâche est appelée périodiquement sous interruption tel que vu en TD), expliquez pourquoi les procédures `mutex_request` et `mutex_release` ci-dessous peuvent ne pas fonctionner normalement à cause d'une interruption.

Question II.2 – 1 point

Protégez les parties critiques du code de `mutex_request` et `mutex_release` au regard d'une commutation de contexte implicite (sous interruption TIMER).

Question II.3 – 1 point

Expliquez comment réaliser une file (FIFO) entre deux tâches en utilisant des mutex. Donnant la déclaration de la `struct FIFO` associée en assumant que la file pourra contenir jusqu'à MAX_FIFO entier (`int`).

Rappel : une file (`struct FIFO`) permet d'ajouter et de retirer des éléments (ici des int) dans la limite d'un buffer. Lorsque le buffer et plein, la tâche qui ajoute un nouvel élément est bloquée, et cela jusqu'à ce qu'une autre tâche vienne retirer un élément. De la même façon une tâche peut retirer des éléments (elle les obtient dans l'ordre dans lequel ils ont été ajoutés, à l'inverse d'une pile) jusqu'à ce que la file soit vide. La tâche est alors bloquée jusqu'à ce qu'une autre tâche ajoute un nouvel élément…

Question II.4 – 1 point

Proposez une implémentation fiable de la fonction C qui lit un entier (`int`) dans la file. Si l'élément n'est pas disponible la fonction bloque la tâche jusqu'à ce qu'un entier y soit ajouté. Le prototype de cette fonction est : `int readByte(struct FIFO*fifo) ;`

Question II.5 – 1 point

Proposez une implémentation fiable de la fonction C qui écrit un entier (int) dans la file. Si la file est pleine la tâche est bloqué est l'entier ajouté une fois que de la place est à nouveau disponible. Le prototype de cette fonction est donné ci-dessous :

```
void writeByte(int value, struct FIFO*fifo) ;
```

Question II.7 – 2 points

Pensant bien faire, un développeur a réutilisé cette implémentation des files pour transmettre dans une file les données reçues depuis un port USB. Lorsqu'une interruption USB signale la présence de données, il écrit ces données dans une file en utilisant la fonction writeByte. Expliquez pourquoi cette implémentation ne fonctionnera pas.

Question II.8 – 1 point

Expliquez, à partir d'un exemple, ce qu'est une situation d'inter blocage lorsque différentes tâches utilisent différents `mutex` tel que définis précédemment. Distinguez deux formes d'interblocages : L'interblocage direct, entre deux tâches, et l'interblocage indirect impliquant plus de deux tâches.

Question II.9 – 1 point

Sur cette base, proposez une fonction C `int directDeadLock(struct mutex_s *mutex)`. Cette fonction permet de détecter un inter blocage direct. Elle retourne 1 lorsque le fait de réserver (via `mutex_request()`) le mutex, pointé par `mutex`, engendre un inter blocage direct et 0 sinon.

Question II.10 – 2 points

Etendez l'algorithme précédent et proposez une fonction C :

`int deadLock(struct mutex_s *mutex)`. Cette fonction permet de détecter un inter blocage direct aussi bien qu'indirect. Elle retourne 1 lorsque le fait de réserver (via `mutex_request()`) le mutex, pointé par `mutex`, engendre un inter blocage. Et 0 si la tâche courante peut réserver le mutex sans se bloquer sans espoir de déblocage.

Partie III. Fiabilité des disques (8 points)

Pour constituer un espace de stockage tolérant aux pannes une solution consiste à combiner plusieurs disques physiques pour constituer un disque virtuel de plus grande capacité. Dans la solution qui nous intéresse, cinq disques sont combinés pour former un disque « virtuel » quatre fois plus grand. Lire un secteur virtuel revient alors à lire le contenu de quatre secteurs physiques et de les délivrer comme un tout. Il faut aussi noter qu'un contrôleur de disque particulier est utilisé pour gérer les 5 disques (1, 2, 3, 4 et 5) utilisés. Il s'agit d'une révision du contrôleur de disque étudié en TD/TP. Les registres d'accès à ce contrôleur de disque sont décrits dans l'extrait ci-dessous :

```
# Configuration des disques durs
# > Contrôleur multi disque
ENABLE_HDM      = 1     # ENABLE_HD=0 => Simulation
                        #    du contrôleur multi-disques désactivée.
HDA_CMDREG      = 0xAF6 # registre de commande du contrôleur
HDA_DATAREGS    = 0xA10 # base des registres de données
                        # (r,r+1,r+2,...r+7)
HDA_IRQ         = 12    # Interruption du contrôleur
```

Extrait du fichier `Hardware.ini`.

L'envoi de commandes se fait également en écrivant sur le port désigné comme port de commande dans le fichier de configuration. Cela permet au microprocesseur de solliciter une opération du disque magnétique. 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. La table 1 (ci-dessous) reprend les principales commandes utiles :

Nom	Code	Port de données (P0; P1; ...; P15)	objet
SEEK	0x02	`diskId` (write int8) ; `numCyl` (write int16) ; `numSec` (write int16).	déplace la tête de lecture du disque `diskId` sur la piste `numCyl`, secteur `numSec`.
READ	0x04	`diskId` (write int8) ; `nbSec` (write int16).	Lit `nbSec` secteurs depuis la position courante du disque `diskId`, et stocke les données lues dans le `MASTERBUFFER`
WRITE	0x06	`diskId` (write int8) ; `nbSec` (write int16).	Ecrit les données contenues dans le `MASTERBUFFER` dans les `nbSec` secteurs depuis la position courante du disque n° `diskId`.
FORMAT	0x08	`diskId` (write int8) ; `nbSec` (write int16) ; `val` (write int32).	initialise `nbSec` secteurs du disque `diskId` avec la valeur `val`.
IRQSEEK	0x12	`diskId` (write int8) ; `numCyl` (write int16) ; `numSec` (write int16).	déplace la tête de lecture du disque `diskId` sur la piste `numCyl`, secteur `numSec`. Cependant, à la différence de la commande SEEK IRQSEEK génère une interruption pour chaque nouveau secteur du disque « survolé » jusqu'à ce qu'elle atteigne `numCyl`, `numSec`.
DSKIRQ	0xA2	`diskId` (read int8) ;	`diskId` indique le numéro du disque qui a générer l'interruption (cette commande retourne une valeur incohérente si aucune interruption n'a été générée).
DSKPOS	0xA4	`diskId` (write int8) ; `numCyl` (read int16) ; `numSec` (read int16).	Retourne le numéro de piste `numCyl` et le numéro de secteur `numSec` associé à la tête de lecture du disk `diskId`.

Table 1 : Commandes ATA-multi-disque

Les données requises dans les registres DATAREGS sont marquées « write » dans la colonne « port de données » et doivent être renseignée par le microprocesseur avant que le registre CMDREG soit programmé avec le code indiqué dans la colonne « Code ». Les données produites en résultats sont marquées « read » et peuvent être consultées une fois le registre CMDREG programmé avec le code indiqué dans la colonne « Code ».

Ce système permet de programmer des opérations « simultanées » sur différents disques. Le numéro du disque « programmé » est donné dans le registre diskId. Ainsi il est par exemple possible de programmer un SEEK sur le disque n°1, puis de lancer la même opération SEEK sur le deuxième disque, sans attendre que le premier et atteint sa destination. Par contre, il est nécessaire d'attendre qu'une commande SEEK, READ, WRITE, ou FORMAT ait généré une interruption sur le disque *n* avant de programmer une nouvelle commande, sur le même disque *n*. Dans le cas contraire, le comportement du disque sollicité est imprédictible.
Cinq MASTERBUFFER nommés MASTERBUFFER1, MASTERBUFFER2, MASTERBUFFER3, MASTERBUFFER4 et MASTERBUFFER5 sont associés aux cinq disques.

Notez encore que lorsqu'une interruption est générée par le contrôleur multidisque (suite à une commande asynchrone SEEK, READ, WRITE, FORMAT ou IRQSEEK), il est nécessaire d'exécuter une commande DSKIRQ pour obtenir, dans le registre de donnée n° 0, le numéro du disque qui a terminé la commande asynchrone.

Le disque surnuméraire est utilisé pour fiabiliser l'assemblage selon les principes des technologies RAID[1]. Le principe retenu en pratique est le suivant, le bit B5 donné dans un secteur du disque surnuméraire vérifie la relation :

$$B1 \oplus B2 \oplus B3 \oplus B4 \oplus B5 = 0 \qquad (0)$$

Où B1, B2, B3, B4 et B5 sont les bits trouvés sur les différents disques à la même position. Cette relation implique qu'en cas de défaillance d'un disque il est possible de reconstruire les données perdues en utilisant les informations correctes stockées sur les autres disques.

Question III.1 – 1 point

Lorsqu'un disque est endommagé, le responsable du matériel remplace le disque et lance un programme buildDisk <n :numero du disque> qui reconstruit le contenu des données stockées sur le disque n.
Donnez l'implémentation de la fonction C sur laquelle repose ce programme de restauration :

```
void computeSector(int size, int *t1, int *t2, int *t3, int *t4, int
*t5) ;
```

Cette fonction prend en paramètre l'adresse de quatre buffer t1, t2, t3 et t4 ou sont stockés les octets des 4 secteurs « corrects » et calcul dans t5 le contenu du secteur de disque perdu.

[1] RAID est l'acronyme de *Redundant Array of Independent Disks*, qu'on peut traduire par « ensemble redondant de disques indépendants »

Question III.2 – 1 point

Expliquez pourquoi un disque virtuel réalisé tel que décrit au début de l'énoncé permet d'écrire quasiment quatre fois plus de données chaque seconde dans un disque virtuel que dans un disque physique de même capacité.

Question III.3 – 1 point

Proposez une implémentation C optimisée (conformément à la question 3) de la procédure :

```
void readVirtualSector(int numCyl, int numSec, char *buffer);
```

Le buffer pointé est 4 fois plus grand que le MASTERBUFFER d'un disque, puisqu'il est destiné à contenir les données issues des 4 disques.

Question III.4 – 1 point

Proposez une implémentation optimisée de la procédure :

```
void writeVirtualSector(int numCyl, int numSec, char *buffer);
```

A l'instar de la procédure élaborée dans la question III.4, la procédure `writeVirtualSector(…)` répartit les données écrites sur les quatre secteurs `numCyl`, `numSec`, des quatre disques dur, 1, 2, 3 et 4. Cependant, afin d'apporter une protection en cas de crash de l'un des cinq disques utilisés, la procédure inscrit, secteur `numCyl`, `numSec` du disque 5, le résultat de la fonction `computeSector` appliqué sur les quatre premiers paquets de données écrits sur les disques 1, 2, 3 et 4.

Question III.5 – 2 points

Une défaillance électrique majeure a engendrée un arrêt des disques lors d'une écriture du disque virtuel. Le système de restauration ne donne pas de bon résultat dans ce cas. Expliquez pourquoi la réponse à la question III.2 est en contradiction avec des écritures fiables dans ce cas ?

Question III.6 – 2 points

Proposez une solution pour garantir la fiabilité des écritures sur le disque virtuel (une écriture sur le disque virtuel doit avoir lieu, ou pas, mais elle ne doit pas laisser les disques physiques dans un état intermédiaire). Décrivez précisément le principe de votre solution, une implémentation C des fonctions de lecture et d'écriture n'est cependant pas demandée.