Maîtrise Informatique
& IUP GMI 3ème année

Le 7 février 2006

Devoir Surveillé
–ASE–
Architecture des Systèmes d'Exploitation

– tous documents autorisés –

Les différences parties peuvent être traitées indépendamment, tous les documents sont autorisés, mais ne perdez pas trop de temps à chercher l'information utile. L'utilisation des calculatrices est conseillée, à défaut donnez les détails du calcul est une estimation du résultat final. Notez encore que l'énoncé contient 23 points, arrêtez-vous à 20 ;) Bonne chance !

Partie I. Gestion– Optimisation du parcours d'un disque (6 points)

Nous nous intéressons ici à la gestion d'un disque aux propriétés particulières. Ce disque dispose de 4096 secteurs et de 8192 pistes. Chaque secteur contient 4096 octets (4 Ko). Ce disque tourne à la vitesse de 7320 tours par minutes, c'est-à-dire 122 tours par seconde ou encore 2 microsecondes pour passer d'un secteur au secteur suivant. La rotation du disque est continue (elle ne s'arrête donc pas même si le disque à déjà atteint le bon secteur mais pas encore la bonne piste par exemple) et le sens de rotation du disque est imposé (la tête de lecture parcourt les secteurs dans l'ordre croissant : 0, 1, 2, …, 4094, 4095, 0,1…). Par ailleurs, la tête de lecture peut se déplacer de la piste 0 à la piste 8191 en 8,192 millisecondes, c'est-à-dire qu'il faut 1 microseconde à la tête de lecture pour avancer ou reculer d'une piste. Le sens du déplacement sur les pistes n'est pas imposé par le matériel.

Les délais de lecture, d'écriture (ou de formatage) sont inscrits dans les délais de « survol » de la piste, et peuvent donc être considérés comme nul par rapport aux délais de déplacement. De plus les fabricants du disque ont organisés la surface du disque de telle sorte que la lecture d'un secteur puisse être enchaînée avec la lecture du même secteur sur la piste suivante (ou précédente) sans pénalité autre que le temps de déplacement. Il en va de même pour les opérations d'écriture et de formatage.

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=0 =>
                        #             simulation du disque désactivée
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
```

La fonction `int _in(int port);` réalise la lecture sur le port désigné. La valeur retournée correspond à l'octet qui a été lu sur ce port. 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 d'un octet sur le port désigné.
L'envoi de commandes se fait également en écrivant sur le port désigné comme port de communication 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 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.
DMASET	0x14	R.F.U.	R.F.U.
DSKNFO	0x16	nbCyl (int16); nbSec (int16); tailleSec (int16)	retourne la géométrie d'un disque
MANUF	0xA2	Id du fabricant du disque (16 octets)	Identifie le disque
DIAG	0xA4	Status	Diagnostic du disque : 0 = KO / 1 = OK

Table 1 : Commandes ATA-2

A l'aide des procédures ci-dessous :

```
void seek(int cyl, int sec) {
   /* set cylinder */
   _out(0x110,(cyl>>8)&0xFF);
   _out(0x111,cyl&0xFF);
   /* set cylinder */
   _out(0x112,(sec>>8)&0xFF);
   _out(0x113,sec&0xFF);
   /* set command */
   _out(0x3F6,0x02);
   /* wait seek */
   _sleep(14);
}
void format_sector(int value) {
   _out(0x110,0);
   _out(0x111,1);
   _out(0x112,(value>>24)&255);
   _out(0x113,(value>>16)&255);
   _out(0x114,(value>>8)&255);
   _out(0x115,(value)&255);
   /* set command format */
   _out(0x3F6,0x08);
   /* wait format */
   _sleep(14);
}
```

Les deux programmes suivants permettent de formater le disque :

Programme 1 :

```
void format(int value) {
 int maxC,maxS;
 int sec,cyl;
 /* Get disk geometry */
 _out(0x3F6,0x16);
 maxC=(_in(0x110)<<8)+_in(0x111);
 maxS=(_in(0x112)<<8)+_in(0x113);
 for(cyl=0;cyl<maxC;cyl++) {
  for(sec=0;sec<maxS;sec++) {
   seek(cyl,sec);    /* seek   */
   format_sector(v); /* format */
}}}
```

Programme 2 :

```
void format(int v) {
 int maxC,maxS;
 int sec,cyl;
 /* Get disk geometry */
 _out(0x3F6,0x16);
 maxC=(_in(0x110)<<8)+_in(0x111);
 maxS=(_in(0x112)<<8)+_in(0x113);
 for(cyl=0;cyl<maxC;cyl++) {
  for(sec=maxS-1;sec>=0;sec--) {
   seek(cyl,sec);    /* seek   */
   format_sector(v); /* format */
}}}
```

Question I.1 – 0,5 point

On considère qu'avant le lancement du formatage la tête de lecture est placée sur le secteur 0 piste 0. Combien de temps prendra le formatage réalisé par le programme 1 ?

Question I.2 – 0,5 point

Selon les mêmes hypothèses combien de temps prendra le formatage réalisé par le programme 2 ?

Question I.3 – 0,5 point

Comme cela est expliqué dans l'énoncé, le fabricant a organisé la surface du disque de manière à permettre certaines optimisations d'accès. Aux vues de ses possibilités jusqu'à combien de secteurs distincts du disque peut-on formater en 4 microsecondes ?

Question I.4 – 0,5 point

En supposant que l'on puisse formater les secteurs du disque à la vitesse annoncée par la question I.3 combien de temps faudrait-il au minimum pour formater tout le disque ?

Question I.5 – 1 point

Proposer une procédure void burstFormatSectors(int sec, int cyl, int value) qui formate en 4 microseconde les n secteurs que vous avez indiqué en réponse à la question I.3 ? Par la suite on considèrera que ces n secteurs forment un « bloc ».

Question I.6 – 1 point

Proposez un algorithme de formatage bas niveau du disque, qui formate la totalité du disque, en le parcourant « bloc » de secteurs par bloc de secteurs (en utilisant la procédure void burstFormatSectors(int sec, int cyl, int value) ;).

Question I.7 – 1 point

Finalement combien de temps ce nouveau programme met-il pour formater le disque ? Pour faire ce calcul, donnez, selon votre algorithme, la liste des 8 couples (secteur/piste) formatés à partir du moment ou le programme formate la position (secteur 4094, piste 0) jusqu'au moment ou il formate la position (secteur 1, piste 2). Expliquez pourquoi vous perdez 8192 microsecondes pendant cet enchaînement de 8 secteurs.

Question I.8 – 1 point

En exploitant le constat de la question I.7, proposez une évolution de votre algorithme (proposer question I.6) dont le temps de formatage soit vraiment proche de la question I.4.

Partie II. Fiabilisation – Intégration de politiques de cache (7 points)

La technologie RAID[1], mise au point par trois chercheurs (Patterson, Gibson et Katz) de l'Université de Californie (Berkeley), permet de constituer un « disque dur virtuel » à partir de plusieurs disques durs physiques. L'intérêt de cette technique est de réaliser un disque qui a soit l'avantage de proposer une grande capacité de stockage accessible à grande vitesse pour un moindre coût, soit d'assurer une grande tolérance aux pannes (haute disponibilité). En effet, la répartition des données sur plusieurs disques durs permet donc d'en augmenter la sécurité et de fiabiliser les services associés.

Les disques durs physiques peuvent être assemblés en un seul disque virtuel selon différentes stratégies proposées par de la technologie RAID. Il existe en tout 7 stratégies différentes que l'on nomme RAID-0, RAID-1, RAID-2… RAID-6 et RAID-7. Chacun d'entre-deux décrit la manière selon laquelle les données sont réparties sur les disques :

- RAID-0: *striping*
- RAID-1: *mirroring*, *shadowing* ou *duplexing*
- RAID-2: *striping* with *parity* (obsolète)
- RAID-3: *disk array with bit-interleaved data*
- RAID-4: *disk array with block-interleaved data*
- RAID-5: *disk array with bit-interleaved distributed parity*
- RAID-6: *disk array with block-interleaved distributed parity*

La technologie RAID-0, que l'on nomme *striping* (traduisez entrelacement) consiste à constitué un disque virtuel en répartissant ses secteurs sur l'ensemble des disques physiques.

Ainsi le secteur 0 du disque virtuel est associé au secteur 0 du disque physique 0, le secteur 1 du disque virtuel est associé au secteur 0 du disque physique 1, le secteur n-1 du disque virtuel est associé au secteur 0 du disque physique n-1, puis, en considérant qu'il existe n disques physiques, le secteur n du disque virtuel sera associé au secteur 1 du disque 0, le secteur 2×n-1 du disque virtuel est associé au secteur 1 du disque n, puis le secteur 2×n du disque virtuel est associé au secteur 2 du disque physique 0… etc

Le disque virtuel bénéficie alors d'une capacité de stockage égale à la somme des capacités de stockage de chaque disque. De plus comme les disques physiques peuvent travailler de manière concurrente (chaque disque de la grappe a son propre contrôleur), le disque virtuel offre une vitesse de transfert élevée. Cependant de cette façon, il n'y a pas de redondance, on ne peut donc pas parler de tolérance aux pannes. En effet en cas de défaillance de l'un des disques, l'intégralité des données réparties sur les disques sera corrompue et donc éventuellement perdue.

La technologie RAID-5 consiste à fiabiliser le fonctionnement du disque virtuel en calculant des bits de parité. Pour chaque tuple de n-1 secteurs (avec n le nombre de disques physiques qui forment le disque virtuel), on écrit un secteur de contrôle qui contient les bits de parité des n-1 secteurs associé. A la différence de la technologie RAID-4, RAID-5 prévoit de ne pas stocker tous les secteurs de contrôle sur le même disque, mais de les répartir sur l'ensemble des disques physiques qui forment le disque virtuel. Le schéma ci-dessous montre une répartition de ce type sur 4 disques.

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

Disque 1	Disque 2	Disque 3	Disque 4
Parité 0+1+2	Secteur 0	Secteur 1	Secteur 2
Secteur 3	Parité 3+4+5	Secteur 4	Secteur 5
Secteur 6	Secteur 7	Parité 6+7+8	Secteur 8
Secteur 9	Secteur 10	Secteur 11	Parité 9+10+11

Table 2 : Répartition des secteurs de parité

Le mode RAID-5 permet d'obtenir des performances très proches de celles obtenues en RAID-0, tout en assurant une tolérance aux pannes élevée, c'est la raison pour laquelle c'est un des modes RAID les plus intéressants en terme de performance et de fiabilité.

Nota : L'espace disque utile sur une grappe de n disques étant égal à n-1 disques, il est intéressant d'avoir un grand nombre de disques pour "rentabiliser" le RAID-5.

Il existe différentes façons de mettre en place une solution RAID sur un serveur :

- de façon matérielle
  - avec des matériels DASD[2] : il s'agit d'unités de stockage externes pourvues d'une alimentation propre. Ce matériel gère lui-même ses disques, de tel sorte que le disque virtuel est reconnu comme un disque physique standard.
  - avec des contrôleurs de disques RAID : il s'agit de cartes s'enfichant dans des slots PCI ou ISA et permettant de contrôler plusieurs disques durs.
- de façon logicielle : il s'agit généralement d'un driver au niveau du système d'exploitation capable de créer un seul volume logique avec plusieurs disques (SCSI ou IDE).

Le but ici est de fournir les fonctions d'accès à un de bloc conformément à la technologie RAID. Les fonctions d'accès à réaliser sont définit ci-dessous :

```
void RAID_read_bloc (int bloc, unsigned char *buffer);
void RAID_write_bloc(int bloc, unsigned char *buffer);
void RAID_format_vol(int bloc, unsigned int value);
```

Ces procédures sont réalisées en utilisant les fonctions « élémentaires » d'accès à un bloc tel que vue en TD/TP lorsqu'il est nécessaire de lire le contenu du volume:

```
void read_bloc  (int vol, int bloc, unsigned char *buffer);
void write_bloc (int vol, int bloc, unsigned char *buffer);
void format_bloc(int vol, int bloc, unsigned int value);
```

On considérera ici que l'on souhaite produire un volume virtuel à partir des volumes physiques 0, 1, 2 et 3 (plutôt qu'un disque virtuelle à partir de plusieurs disques physiques). Pour simplifier l'énoncé on supposera aussi que les 4 volumes ont la même taille (sont composées du même nombre de blocs) et que ce nombre est de NB_BLOCS. Enfin les blocs de chaque partition on la même taille BLOC_SIZE (en octets).

Dans un premier temps, on s'intéresse à la technologie RAID-0.

Question II.1 – 0,5 point

Proposez une implémentation des trois procédures d'accès RAID au données (`RAID_read_bloc(…), RAID_write_bloc(…), RAID_format_bloc(…)`) conformément à la technologie RAID-0.

[2] DASD : *Direct Access Stockage Device* à traduire par Support de stockage à accès direct

Question II.2 – 1 point

Expliquez pourquoi et comment les procédures d'accès au volume virtuel peuvent être plus performantes (en temps d'accès) que les procédures d'accès aux volumes réels.

Nous nous intéressons maintenant à la technologie RAID-5.

Question II.3 – 0,5 point

Proposez une implémentation de la fonction qui calcule un octet de parité à partir de 3 octets de données, de tel sorte que le premier bit de l'octet de parité assure la parité paire du nombre de 1 du premier bit des 3 octets de données, le 2ème bit de l'octet de parité assure la parité paire du nombre de 1 du 2ème bit des 3 octets de données, …etc.

La signature de cette procédure est :

```
unsigned char byteParity(unsigned char[3]) ;
```

Question II.4 – 0,5 point

A partir de la fonction définit dans la question 4, proposez une procédure qui calcule le contenu du secteur de parité (`blocP`) à partir du contenu des 3 autres blocs (`bloc0`, `bloc1`, `bloc2`). Cette procédure est définit par la signature suivante :

```
void blocParity( unsigned char *bloc0, unsigned char *bloc1,
                 unsigned char *bloc2, unsigned char *blocP);
```

Question II.5 – 1 point

Pour respecter le principe de répartition des blocs de parité tel qu'il est montré par la Table 2, proposez une implémentation des 4 fonctions ci-dessous :

- Une fonction `int vol4vbloc(int vbloc);` qui calcule le numéro de volume qui contiendra le bloc n du disque virtuel.
- Une fonction `int bloc4vbloc(int vbloc);` qui calcule, à partir du bloc virtuel vbloc le numéro du bloc « réel » au sein du volume désigné par vol4vbloc(…).
- Une fonction `int vol4blocp(int vbloc);` qui calcule, à partir d'un numéro de bloc virtuel le numéro du volume qui contient le bloc de parité associé.
- Une fonction `int bloc4blocp(int vbloc);` qui calcule, à partir d'un numéro de bloc virtuel le numéro du bloc de parité au sein du volume désigné par vol4blocp(…).

Question II.6 – 0,5 point

A l'aide des fonctions de la question II.5, proposez une implémentation de la procédure :

```
void RAID_read_bloc(int bloc, unsigned char *buffer);
```

Question II.7 – 1 point

A l'aide des fonctions de la question II.4 et II.5, proposez une implémentation de la procédure :

```
void RAID_write_bloc(int bloc, unsigned char *buffer);
```

Question II.8 – 1 point

L'un des bénéfices de la technologie RAID-5 est de permettre la reconstruction d'un disque détruit à partir des disques existants. Expliquez comment on peut reconstruire un bloc de donnée à partir des deux autres blocs et d'un bloc de parité ? Généralisez en proposant la reconstruction de n'importe quel bloc.

Question II.9 – 1 point

Proposez une procédure capable de reconstruire un bloc (`bloc3`) détruit à partir de 3 blocs (`bloc0` et `bloc1` et `bloc2`). Cette procédure est définit de la manière suivante :

```
void extractBloc( unsigned char *bloc0, unsigned char *bloc1,
                  unsigned char *bloc2, unsigned char *bloc3);
```

III. Abstraction – Systèmes multi-microprocesseurs (7 points)

Lorsqu'un système informatique dispose de plusieurs microprocesseurs qui partagent la même mémoire physique, un circuit matériel est nécessaire pour mettre en œuvre un mécanisme d'exclusion mutuelle (qui permet de garantir qu'une section de code ne sera exécutée que par un seul microprocesseur à la fois). Dans le système informatique considéré ici, l'exclusion mutuelle est assurée par un circuit extérieur aux deux microprocesseurs accessible via les port 0xA0, 0xA1, 0xA2, … 0xAF. Chaque port contient un octet initialisé à 0xFF. Il est possible d'obtenir l'état du verrou en lisant le port associé via la fonction : `int _in(int port) ;`
Le fait qu'un programme exécuté par un microprocesseur lise un verrou fait passer la valeur du verrou de `0xFF` à `0x0`. Ainsi un premier appel à `_in(0xA0);` retourne `0xFF` mais ensuite le verrou passe à `0x0` et tout nouvel appel à `_in(0xA0);` retourne `0x00`. De plus si les deux microprocesseurs appellent simultanément `_in(0xA0);` un seul des deux reçoit la valeur `0xFF`, l'autre reçoit la valeur 0x0. C'est grâce à ce dernier point qu'il devient possible de « départager » deux demandes concurrentes d'exclusions mutuelles. Il est ensuite possible de rendre un verrou en écrivant `0xFF` dans le port associé via la fonction : `void _out(int port, int value) ;`
Grâce à ce matériel il devient possible de réaliser toutes les opérations de synchronisation de processus imaginables sur un système multiprocesseur.

Par ailleurs le circuit associé au port `0xB0` des deux microprocesseurs permet de déterminer quel microprocesseur exécute une section de code. Ainsi, si le processeur 1 lit le port `0xB0` (en exécutant `_in(0xB0);`) la valeur lue sera 0, alors que si le processeur 2 lit le même port `0xB0`, la valeur lue sera 1.

Enfin le système multiprocesseur considéré ici dispose d'autant de circuit timer que de microprocesseur. Ainsi lorsqu'un programme exécuté par un microprocesseur configure une interruption basée sur le temps (dans le cas d'un ordonnanceur par exemple), cette interruption ne sera signalée qu'au seul processeur qui l'a réclamée. ***Notez de plus***, que lorsqu'un microprocesseur exécute un `irq_disable();` pour stopper la réception des interruptions ou un `irq_enable();` pour l'autoriser cela n'a aucune influence sur les interruptions adressées à d'autres microprocesseurs.

Question III.1 – 0,5 point

Expliquez comment réaliser un mécanisme d'exclusion mutuelle entre deux programmes tournant chacun sur un processeur à l'aide le circuit d'exclusion mutuelle décrit précédemment. (On considère pour l'instant qu'il n'y a pas de système d'ordonnancement de tâche au sein de chaque processeur).

Question III.2 – 1 point

Donnez une implémentation des procédures :
`void lock(int numlock);` et `void unlock(int numlock);`
de tel sorte que si deux processeurs exécutent chacun un programme qui appelle `lock(2);` (par exemple), l'une des deux exécutions du programme (sur l'un des deux processeur) reste « bloquée » (en attente active) dans l'appel de la procédure `lock(2);` jusqu'à ce que l'autre appelle `unlock(2);` (sur 2, c'est-à-dire sur le même verrou). Ici `numlock` correspond simplement à un numéro entre 0 et 15 qui définit lequel des verrous matériels est utilisé.

Question III.3 – 0,5 point

Expliquez en quelques phrases comment les procédures qui assurent la commutation de tâches définies ci-dessous (vues en TD) peuvent être modifiées pour qu'elles soient utilisables sur un matériel multiprocesseur, c'est-à-dire pour qu'elles soient utilisées par les deux microprocesseurs pour ordonnancer un jeu de tâches « communes ».

```
struct ctx_s *ring;                    /* ring of unterminated ctx */

void yield() {
  while (ring->state==Blocked) {       /* excepted if it's Blocked */
     irq_disable();
     ring = ring->next;                /* so try the next one      */
     irq_enable();
  }
  switchToCtx(ring);                   /* and switch on it!        */
}
void switchToCtx(struct CPUCtx *newCtx) {
  irq_disable();                      /* begin of a critical path  */
  ASSERT(newCtx->magic!=MAGICWORD);   /* the ctx must be created   */
  ASSERT(newCtx->status!=Terminated); /* and not terminated.       */
                                      /* save currentCtx.          */
  asm(" mov %%esp,%0" : "=r" (currentCtx->savedESP) );
  asm(" mov %%ebp,%0" : "=r" (currentCtx->savedEBP) );
  currentCtx = newCtx;
                                      /* and restore the newCtx    */
  asm(" mov %0,%%esp" : : "r" (currentCtx->savedESP) );
  asm(" mov %0,%%ebp" : : "r" (currentCtx->savedEBP) );
  if(currentCtx->status == Ready){    /* is it a new task ?        */
    irq_enable();                     /* end of a critical path    */
    startCurrentCtx();                /* then call it...           */
  } else {
    irq_enable();                     /* end of a critical path    */
    return ;                       /* else restore the caller frame */
  }
}
```

Question III.4 – 1 point

Proposez une évolution des procédures C :
`void yield();` et `void switchCtx(struct ctx_s newCtx);`
(ainsi que des variables globales qu'elles utilisent) pour qu'elles fonctionnent en contexte multiprocesseur. Vous pourrez utiliser le port `0xB0` pour connaître le numéro du processeur qui exécute le code et les procédures `void lock(int numlock)` et `void unlock(int numLock)` précédemment décrites pour protéger les sections de codes qui selon vous doivent l'être pour que l'ordonnanceur fonctionne s'il est exécuté par les deux microprocesseurs simultanément.

Question III.5 – 1 point

En considérant maintenant que le mécanisme d'ordonnancement décrit précédemment opère sur chaque microprocesseur : Expliquez, en quelques lignes, pourquoi le mécanisme d'exclusion mutuelle (ci-dessous) qui fonctionne correctement sur un système monoprocesseur ne fonctionne plus sur le système multiprocesseur.

```
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;
```

```
...
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 */
  irq_disable();                /* begin of a critical path        */
  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;
    irq_enable();               /* end of the critical path        */
    yield();                    /* schedule another task           */
  } else {                      /* else, if the mutex is available */
    mutex->locked = 1;          /* the mutex is no more 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;
    irq_enable();               /* end of the critical path        */
  }
}
void mutex_release(struct mutex_s *mutex) {      /* unlock a mutex */
  irq_disable();                /* begin of a critical path        */
  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->taskList) {       /* and (at less) a task is locked  */
      if(!mutex->taskList)  /* if nothing is waiting for the mutex */
        mutex->locked = 0;      /* the mutex is now unlocked       */
      else {                    /* else if something is pending    */
        mutex->taskList=Ready;  /* we can unlock a locked task     */
        mutex->owner=mutex->taskList;/* it is the new mutex owner. */
                   /* the next task is now the first waiting task */
        mutex->taskList=mutex->taskList->taskList;
      }
    }
  }
  irq_enable();                 /* end of the critical path        */
}
```

Question III.6 – 1 point

Proposer une modification de sem_up et sem_down pour que le mécanisme de sémaphore soit opérationnel sur un système bi-processeur.

Question III.7 – 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.

Question III.8 – 1 point

Sur cette base, proposez une fonction C `int deadLock(struct mutex_s *mutex)` qui retourne 1 si le fait que le contexte courant (pointé par la variable globale déjà déclarée `currentCtx`) réserve le mutex pointé par `mutex` engendra un inter blocage de certains contextes, et 0 sinon.