Architecture des Systèmes d'Exploitation
Maîtrise d'Informatique

Université de Lille 1 SCIENCES ET TECHNOLOGIES

Janvier 2016

Devoir Surveillé

– tous documents autorisés –

Le langage C est requis pour les implémentations

Pensez à détailler vos raisonnements et vos calculs

Tous les documents sont autorisés, mais ne perdez pas trop de temps à chercher l'information utile. ***La troisième partie (page 5/6) est un QCM à joindre à votre copie.*** *Les questions sont indépendantes. La première et la seconde partie constituent deux exercices indépendants. Cependant au sein de chaque exercice les questions sont organisées dans un ordre logique. Notez encore que l'énoncé contient 24 points, je m'arrêterai à 20/20 ;-)*

Exercice I. Volumes RAID (8 points)

Nous nous intéressons ici à l'utilisation de la technologie RAID : Redundant Array of Independent Disks, tel qu'elle a été présentée en cours. Cette technologie permet d'assurer la fiabilité de système de stockage en utilisant simultanément plusieurs volumes distincts. La technologie RAID se décline en plusieurs versions : RAID0, RAID1, RAID2... RAID6.

La technologie RAID1 par exemple, consiste simplement à utiliser 2 volumes (ou plus), de tailles identiques, idéalement situés sur des disques physiques différents. Les 2 volumes sont utilisés pour former un volume RAID1 (virtuel). Lorsqu'une écriture est demandée sur le volume virtuel RAID1 elle est en fait réalisée, simultanément, sur les deux volumes qui lui sont associés. Ainsi, à chaque instant, si l'un des volumes sous-jacent crashe, l'administrateur du système possède une « copie » : l'autre volume.

Question I.1 (1 point)

Nous considérons ici l'utilisation de disque dur mécaniques, par de SSD. Lorsque la technologie RAID1 est utilisée pour former un volume virtuel à partir de deux volumes issus d'un même disque on observe que le volume RAID est bien plus lent que le disque physique. Pour améliorer les performances, il est conseillé d'utiliser deux disques distincts. Pourquoi, selon vous, est-il conseillé que les volumes physiques qui composent le volume RAID1 virtuel soient sur des disques différents ?

Question I.2 (1 point)

Nous considérons donc que deux disques mécaniques distincts sont utilisés, mais aussi qu'ils ont la même vitesse de rotation. S'il faut n secondes pour entièrement effacer le contenu de l'un des volumes physiques. Combien de secondes seront nécessaires, dans le meilleur des cas, pour entièrement effacer le contenu d'un volume virtuel RAID1 ayant la même capacité de stockage ?

Question I.3 (1 point)

Nous considérons maintenant l'utilisation de disques SSD. Pensez-vous que dans ce cas, il est toujours préférable, pour des raisons de performances, d'utiliser deux disques distincts pour stocker les deux volumes qui compose le volume RAID1. Expliquez votre raisonnement.

Question I.4 (1 point)

Nous nous proposons d'implémenter simplement la technologie RAID1 sur la base de code utilisée en TP2. Vous pouvez donc considérez que vous disposez des fonctions d'accès à un volume écrites en TP :

```
void readbloc(int vol, int bloc, char * buffer);
void writebloc(int vol, int bloc, char *buffer);
```

Nous assumons ici que les volumes peuvent être associés à des disques différents et que l'implémentation des fonctions a donc été modifiée en conséquence.

Les volumes physiques à utiliser pour réaliser le volume virtuel RAID1 sont définit par les deux variables globales suivante :

```
int RAID1_vol1 ;
int RAID1_vol2 ;
```

Proposez une implémentation simple des fonctions de lecture et d'écriture sur le volume RAID1. Pour cela vous utiliserez les fonctions readbloc et writebloc. Les prototypes des fonctions à implémenter sont les suivants :

```
void RAID1_writebloc(int bloc, char *buffer) ;
void RAID1_readbloc(int bloc, char *buffer) ;
```

Question I.5 (1 point)

Le problème de cette première implémentation est que même si les volumes sollicités sont associés à des disques différents, la fonction `RAID1_writebloc` est au moins deux fois plus lente qu'un simple `writebloc`... Pour permettre une implémentation plus efficace, l'API d'accès aux volumes est modifiée comme suit `writebloc` est remplacé par deux fonctions :

```
void request_writebloc(int vol, int bloc, char *buffer) ;
void wait_completion(int vol);
```

La première fonction demande l'écriture d'un bloc sur un volume (du disque associé au volume). Mais elle n'attend pas que cette opération soit réalisée sur le disque.
La seconde fonction elle, permet d'attendre que la commande d'écriture en court, sur le volume vol soit terminée.

Proposez une nouvelle implémentation de la fonction `RAID1_writebloc` qui utilise cette API et qui soit au moins deux fois plus performante que celle de la question précédente.

Question I.6 (1 point)

En l'état, la fonction `RAID1_readbloc` est toujours contre performante. Une évolution de l'API de gestion des volumes et proposée pour vous permettre d'implémenter une version performante de la technologie RAID1.

```
int isVolumeBusy(int vol) ;
void readbloc(int vol, int bloc, char *buffer) ;
```

La première fonction permet de déterminer si le volume vol est actuellement occupé (par une requête d'écriture ou de lecture).

La seconde fonction a le même comportement que celui vu en TD/TP.

Proposez une implémentation plus efficace de la fonction `RAID1_readbloc` en utilisant cette nouvelle API d'accès aux volumes.

Exercice II. Rendez-vous (6 points)

La primitive de rendez-vous est utilisée lorsqu'un programme implique plusieurs contexte d'exécution (tel que vu dans le premier TP) qui doivent tous avoir atteint une certaine étape, le « rendez-vous », avant de reprendre leur exécution. L'API d'un service de rendez-vous que nous considérons est la suivante :

```
1.   /* La structure RendezVous_s permet de définir un rendez-
     vous */
2.   struct RendezVous_s;
3.   /* prepareRDV permet d'initialiser une variable du type
4.      struct RendezVous_s en indiquant le nombre de contextes
5.      d'exécutions attendus. */
6.   void prepareRDV(
7.     struct RendezVous_s *RendezVous,
8.     int nbProcess);
9.
10.   /* attendreRDV permet à un contexte d'être bloqué jusqu'à
11.      ce que tous les contextes d'exécutions attendus aient
12.      appelés attendreRDV */
13.   void attendreRDV(struct RendezVous_s *RendezVous) ;
```

Les rendez-vous sont généralement utilisés pour « paralléliser » des traitements complexes. L'idée est alors de décomposer le traitement long en n traitements plus court, puis de lancer les n fonctions associées dans des contextes disjoints (et si possible sur des processeurs différents) puis d'attendre un rendez-vous. Lorsque les n contextes ont atteint le rendez-vous c'est que les n parties du traitement ont été exécutées en parallèle (donc jusqu'à n fois plus vite s'il y a n microprocesseurs).
L'utilisation des RendezVous tel que présentée si dessus est liée à l'implémentation des contextes vus durant le premier TP. Pour mémoire l'API des contextes était la suivante :

```
1. #define CTX_MAGIC 0xCAD0AB0B
2. typedef void (t_fct) (void *);
3.
4. enum ctx_state_e { /* enum des etats d'un contexte     */
5.    CTX_READY,      /*  . Contexte activable           */
6.    CTX_ACTIVABLE,  /*  . Contexte re-activable        */
7.    CTX_TERMINATED  /*  . Contexte termine             */
8. };
9. struct s_ctx {
10.     unsigned int     magic ;     /* detrompeur            */
11.     void            *savedESP ; /* valeur courante d'ESP */
12.     void            *savedEBP ; /* valeur courante d'EBP */
13.     t_fct           *startfct ; /* fonction de depart    */
14.     void            *arg ;      /* argument de depart    */
15.     char            *stack ;    /* pile d'execution      */
16.     enum ctx_state_e state ;    /* etat courant          */
17.     struct s_ctx    *nextCtx ;  /* chaînage pour "ring"  */
18.   } ;
19.   /* anneau des contextes ordonnancables               */
```

```
20.   struct ctx_s * ring      = NULL ;
21.   /* contexte d'exécution actuellement actif              */
22.   struct s_ctx *currentCtx = NULL ;
23.   /* Creation d'un nouveau contexte à ordonnancer         */
24.   int initCtx(struct s_ctx *aCtx, int stackSize, t_fct f,
      void *arg) ;
25.   /* Implementation de la strategie d'ordonnancement      */
26.   void yield() ;
27.   /* Implem. du mecanisme de commutation de contexte      */
28.   void switchToCtx(struct s_ctx *aCtx) ;
29.   /* demarrage de la préemption de tâche                  */
30.   void setupCtx() ;
31.   /* structure d'une variable de type semaphore           */
32.   struct s_sem ;
33.   /* initialisation d'une semaphore sem à la valeur v     */
34.   void sem_init(struct s_sem *s, int v);
35.   /* liberation d'une semaphore. Cf TP.                   */
36.   void sem_up(struct s_sem *sem);
37.   /* reservation d'une semaphore. Cf TP.                  */
38.   void sem_down(struct s_sem *sem);
```

listing Ctx.h

Question II.1 (2 points)

Nous nous proposons dans un premier temps de réaliser une fonction de tri accéléré parallèle :

```
void psort(int n, int *outarray, int*inarray);
```

cette fonction de tri découpe le tableau inarray en n tableaux, et exécute n appels à l'algorithme de tri `void qsort(int *outarray, int *inarray) ;` (quick sort) dans n contextes différents (en parallèle donc, si l'ordonnancement gère plusieurs microprocesseurs).
Une fois chacun des n fragments de tableau trié par chacun des n contextes, l'ensemble des sous-tableaux sont fusionnés en un seul tableau trié grâce à la fonction

```
void fusion(int n, int *outarray, int *inarray1,...) ;
```

Implémentez la fonction psort en utilisant les fonctions qsort et fusion, mais aussi les fonctions prepareRDV et attendreRDV pour gérer le parallélisme.

Question II.2 (2 points)

Le mécansime de rendez-vous peut facilement être implémenté avec des sémaphores. Proposez une implémentation de la structure s_rendezVous ; ainsi que des fonctions prepareRDV et attendreRDV, en utilisant l'API des sémaphores implémentée en TP est dont le prototype des fonctions est rappelé au début de l'énoncé.

Question II.3 (2 points)

Selon l'implémentation qui a été faite en TP, de votre ordonnanceur, pensez-vous que psort sera plus rapide que qsort ou moins rapide ? Pourquoi ?

Master 1 – mention Informatique
Composition de : ASE -
Architecture des Systèmes d'Exploitation

N° PLACE →

Attention : N'oubliez pas de joindre ce volet du sujet à votre copie

III. Questions à Choix Multiples (12 points)

Les six questions suivantes sont des contrôles de connaissance sous la forme d'un QCM. Chaque question peut avoir entre une et quatre réponses valides. Cochez les réponses valides et joignez la page du sujet à votre copie. Pour une question donnée (il y en a 6 dans cette partie), chaque choix correct vaut ½ point, chaque erreur vous fait perdre ½ point. Donc, si vous sélectionnez l'ensemble des bons choix votre note pour cette question sera de 2. Si tous vos choix sont faux, la note de la question sera -2. Aussi, pour éviter une note négative à une question, vous pouvez cocher la case « je ne sais pas » votre note pour cette question sera alors de 0 (quel que soit les autres choix cochés).Globalement, si la note résultant de ce QCM est négative, elle sera ignorée.

Question III.1 (2 points)

Le laps de temps entre deux changements de contexte est, dans un système d'exploitation contemporain, d'une milliseconde. Le choix d'un laps de temps plus important …

- n'a aucune conséquence. ☐
- est toujours déconseillé. ☐
- peut rendre moins réactive les interactions avec l'utilisateur. ☐
- est conseillé pour les systèmes supportant des serveurs web. ☐
- Je ne sais pas ☐

Question III.2 (2 points)

Les sémaphores peuvent notamment être utilisés pour …

- s'assurer que des procédures sont exécutées en exclusion mutuelle. ☐
- éviter les inter-blocages entre processus. ☐
- organiser des rendez-vous entre processus. ☐
- faire, autrement, tous ce que les moniteurs de Hoare peuvent faire. ☐
- Je ne sais pas ☐

Question III.3 (2 points)

La mémoire virtuelle permet …

- d'accélérer les accès à la mémoire physique. ☐
- d'éviter que le crash d'un programme n'entraine le crash l'ordinateur. ☐
- de donner l'illusion que l'on a plus de RAM que l'on en a réellement. ☐
- de protéger les données critiques du système d'exploitation. ☐
- Je ne sais pas ☐

Question III.4 (2 points)

La mémoire virtuelle des dernières générations de processeurs Intel définit un espace adressable de 48 bits. En conséquence

- un processus adresse potentiellement 256To de mémoire virtuelle. ☐
- 256 processus utilisent chacun au maximum 1To de mémoire virtuelle. ☐
- la mémoire physique adressable est au maximum de 256To. ☐
- la mémoire physique est au maximum de 6 octets. ☐
- Je ne sais pas ☐

Question III.5 (2 points)

Sur un même disque mécanique, on observe le temps que l'on met à lire n octets séquentiels dans un fichier A par rapport au temps que l'on met à lire n octets séquentiels dans un fichier B.

- Lire n octets dans A prendra toujours exactement le même temps. ☐
- Pour lire n octets on observera le même temps quel que soit le fichier. ☐
- Si B est plus fragmenté que A, le temps de lecture sur B sera meilleur. ☐
- Si les données de B sont stockées au centre du disque et celles de A à la périphérie, les n octets seront lus plus lentement sur B. ☐
- Je ne sais pas ☐

Question III.6 (2 points)

Un serveur NAS professionnel utilise la technologie RAID 51 avec 6 disques de 2To. 2 grappes de disques sont utilisées en RAID 1 (*mirroring*). Chaque grappe, composée de 3 disques, utilise un RAID5 (*striping*).

- Après initialisation le serveur NAS dispose donc 12To d'espace libre. ☐
- Les fichiers sont préservés même si 2 disques quelconques cassent. ☐
- Les fichiers sont préservés même si 3 disques quelconques cassent. ☐
- Les fichiers sont préservés même si 4 disques quelconques cassent. ☐
- Je ne sais pas ☐