Examen (1ère session) – 16 décembre 2014

N. Sabouret

L'épreuve dure 2h30. Tous les documents sont autorisés. Les exercices sont indépendants.

1 Exercice 1 – Question de cours (4 points)

1. Quel est l'intérêt de la pagination à deux niveaux ? Expliquez (1 point)

   Réduire la taille mémoire des tables de pages, lorsque toutes les pages ne sont pas utilisées.

2. À quoi sert l'ouverture d'un fichier ? (1 point)

   À charger le FCB à partir du répertoire dans la RAM.

3. Quel est l'algorithme d'ordonnencement de processus le plus efficace ? Justifiez. (1 point)

   Aucun, cela dépend des processus effectivement en attente

4. Quel mécanisme permet d'éviter les attentes actives dans les méthodes de synchronisation de processus dans un OS ? (1 point)

   Les appels systèmes

2 Exercice 2 – Gestion de la mémoire (5 points)

On considère un système de gestion de mémoire de 16 Ko, gérée de manière segmentée et paginée avec un seul niveau de pagination. Un processus peut avoir au plus 4 segments. Chaque segment peut adresser au plus 1 ko de données ou de code. Enfin, la taille des cadres de page est fixée à 256 octets.

1. Quelle est la taille (en nombre de bits) et la composition de l'adresse physique? Expliquez. (0,5 point)

   Si on a 16Ko de RAM, l'adresse physique est forcément sur 14 bits, avec 8 bits pour le décallage (256o par cadre de page) et 6 bits pour le numéro de cadre (64 cadres en tout).

2. Quelle est la taille (en nombre de bits) de l'adresse logique? Expliquez. (0,5 point)

   Pour adresser 4 segments par processus, il faut 2 bits. Donc le sélecteur dans l'adresse logique est sur 2 bits. Si chaque segment peut adresser au plus 1 Ko de données, le décalage est sur 10 bits. L'adresse logique est donc sur 12 bits (chaque processus adresse au plus 4 Ko).

3. Quelle est la taille (en nombre de bits) de l'adresse linéaire? Expliquez. (0,5 point)

   Comme pour l'adresse physique, on aura 8 bits pour le décallage. Par contre, on n'a pas besoin de 2^6 pages puisque chaque processus peut adresser au plus 4Ko, il suffit de 16 pages de 256o, et donc le numéro de page tient sur 4 bits. L'adresse linéaire est donc sur 12 bits.

4. Quelle est la taille de chaque table des pages et de la table des segments? Expliquez. (1,5 point)

   *Chaque table des pages contient 2^4 lignes de 6 bits (numéro du cadre) + 1 bit de validité. Soit un total de 16*7=112 bits (14 octets).*

   *La table des segments contient **au plus** 4 lignes de 10 (limite) + 12 bits (base) chacune, soit en tout 88 bits (11 octets).*

5. On considère l'adresse logique `9A1` (exprimée en hexadécimal, c'est-à-dire 1001 1010 0001). Expliquez comment est calculée l'adresse physique et donnez sa valeur en indiquant clairement tous les calculs intermédiaires (2 points).

   On suppose que le processus utilise 4 segments et que la table des segments contient les valeurs suivantes (toutes données en hexadécimal):

Segment:	Limite	Base
0	24A	D29
1	0B7	127
2	1C8	B32
3	037	086

   On suppose aussi que la table des pages du processus est la suivante:

Page:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Cadre:	0E	17	01	2A	20	18	30	23	00	00	12	3E	05	00	04	37
Valide:	1	1	1	1	0	1	0	1	1	0	0	1	1	0	0	1

   Adresse logique: 9A1, se décompose en numéro de segment (2) et décalage (1A1)

   La table des segments indique que le segment 2 a une limite de 1C8 qui est bien supérieure au décalage.

   L'adresse linéaire est donc B32 (base) + 1A1 (décalage) = CD3 (1100 1101 0011 sur 12 bits), ce qui se décompose en C pour le numéro de page et D3 pour le décalage.

   D'après la table des pages, la page C est située sur le cadre 5 (00 0100).

   L'adresse physique est donc 00 0100 1101 0011, c'est-à-dire 05 D3

3 Exercice 3 – Disques RAID (3 points)

Soit les données hexadécimales suivantes:

```
FF 16 12 39 A2 27 47 17 BD E3 89 62 38 C4 8C
```

que l'on souhaite écrire sur un disque RAID constitué de 4 disques et organisé en agrégat par bandes avec un disque de parité (RAID 5). On suppose que les bandes sont constituées d'un seul octet par disque, que la première bande utilise le quatrième disque pour stocker la parité des trois premiers octets, la deuxième bande verra sa parité enregistrée sur le premier disque, la troisième bande sur le deuxiéme disque, etc.

1. Quelle est la capacité totale de ce sous-système RAID5 sachant que chaque disque comporte 20Go d'espace disponible? Justifiez. (0,5 point)

   La capacité totale du système RAID5 est $3 \times 20Go$, soit $60Go$. L'un des disque est globalement utilisé pour stocker la parité

2. Calculer la parité associée à chaque bande et compléter le tableau suivant (1,5 points). Vous pouvez rajouter des bandes si nécessaire.

	Disque 0	Disque 1	Disque 2	Disque 3
Bande 0				
Bande 1				
Bande 2				
...				

```
          Disque0 - Disque1 - Disque2 - Disque3
Bande 0      FF        16        12       *FB*
Bande 1     *BC*       39        A2        27
Bande 2      47       *ED*       17        BD
Bande 3      E3        89       *08*       62
Bande 4      38        C4        8C       *70*
```

3. Supposons que le disque 1 est tombé en panne. Comment faire pour recalculer les quatre parties de bandes effacées de ce disque ? Expliquez. (1 point)

En cas de défaillance d'un disque, les données qui s'y trouvaient pourront être reconstituées par l'opération XOR. En effet, l'opération XOR a la propriété suivante : si on considère blocs de taille identique $a_1, a_2, ..., a_n$ et si $a_1 \times a_2 ... \times a_n = p_1$ alors $p1 \times a1 \times a2 ... \times a_{n-1} = a_n$.

4 Exercice 4 – Ordonnencement de disques (5 points)

On considère un disque composé de 256 cylindres (numérotés de 0 à 255) recevant des requêtes d'accès à des blocs situés sur des cylindres différents. On s'intéresse ici uniquement aux cylindres. On suppose que le déplacement de la tête d'un cylindre à l'autre prend 1 unité de temps. On suppose que la vitesse de rotation du cylindre est de 10 unités de temps et, par conséquent, que le traitement d'une requête prend toujours 10 unités de temps une fois la tête positionnée. La table ci-dessous décrit les arrivées de requêtes sur le contrôleur de disque:

Date	Cylindre(s)
0	167, 21
53	213, 98
135	87
215	4, 8, 131
523	213, 81

Initialement, la tête est positionnée sur le cylindre 37 et il n'y a aucune requête dans la file d'attente.

Important: une requête de déplacement du bras ne peut pas être interrompue: si une nouvelle requête plus prioritaire arrive pendant que le bras se déplace pour atteindre la prochaine requête, elle sera traitée ultérieurement.

1. Décrivez l'exécution si le contrôleur utilise un algorithme FCFS (1 point) et donnez le temps d'exécution total (0,5 point).

   Dans le tableau suivant, les dates sont indiquées en fin de lecture du cylindre:

Date	*0*	*140*	*296*	*498*	*623*	*644*	*737*	*751*	*884*	*976*	*1118*
Cyl.	*37*	*167*	*21*	*213*	*98*	*87*	*4*	*8*	*131*	*213*	*81*

2. Décrivez l'exécution si le contrôleur utilise un algorithme SSTF (1 point) et donnez le temps d'exécution total (0,5 point).

Date	*0*	*26*	*182*	*238*	*330*	*373*	*394*	*483*	*497*	*523*	*610*	*752*
Cyl.	*37*	*21*	*167*	*213*	*131*	*98*	*87*	*8*	*4*	*4*	*81*	*213*

Au temps 26, nous n'avons pas reçu les suivants donc nous allons au 167.

Au temps 182, nous avons reçu 87, 98 et 213. SSTF choisit 213 qui est la plus proche.

Au temps 238, nous avons reçu 4, 8, 87, 98, 131. SSTF choisit 131 qui est la plus proche, puis les suivantes dans l'ordre décroissant, qui seront toutes traitées avant la date 523 de la nouvelle série de requêtes

Au temps 497, nous n'avons plus rien en stock, donc nous sommes attendons le temps 523, où nous traitons 81 puis 213.

3. Décrivez l'exécution si le contrôleur utilise un algorithme C-LOOK (1,5 point) et donnez le temps d'exécution total (0,5 point). On suppose que la tête de lecture est montante dans l'état initial.

Date	*0*	*140*	*196*	*227*	*303*	*324*	*367*	*497*	*511*	*523*	*606*	*748*
Cyl.	*37*	*167*	*213*	*21*	*87*	*98*	*131*	*4*	*8*	*8*	*81*	*213*

Au départ, on va vers 167. À 140, on a dans la file 21, 87, 98 et 213 donc on continue vers 213.

À 196, on n'a plus rien en montant, donc on repart d'en bas (21). 213 est à 42 du bord, donc il faut encore se déplacer de (42-21)=21 avant de lire

À 227, on a dans la file 4, 8 (dépassés), 87, 98, 131. On continue en montant jusqu'à 131.

À 367, il reste 4 et 8 dans la file. Nous n'avons pas encore reçu les requêtes 213 et 81. Donc nous allons jusqu'à 4, c'est-à-dire par symétrie 255-4=251. Il faut donc avancer de 120 pas.

Au temps 511, nous devons attendre la prochaine requête, que nous traitons dans l'ordre montant.

5 Exercice 5 – Processus parallèles (3 points)

On s'intéresse à des séquences d'opérations et on souhaite définir une méthode pour notre système d'exploitation qui parallélise les calculs dans des processus différents lorsque c'est possible. Par exemple, si on calcule successivement:

```
op1 : c = a+b
op2 : d = a-b
op3 : a = c-d
```

Les deux premières opérations sont parallélisables, mais la troisième a besoin du résultat des deux premières pour s'exécuter.

Considérons la séquence d'opérations suivante (on suppose que les variables x, y et z ont été définies avant):

```
op1 : a = x + y
op2 : b = x * y
op3 : c = a + z
op4 : d = a * b
op5 : e = c / d
```

1. Dessinez sous la forme d'un graphe orienté les dépendences entre les opérations. Les nœuds du graphes sont les opérations et les arètes décrivent des dépendances. (1 point)
   À dessiner: 1<3, 1<4 2<4, 3<5, 4<5

2. Cette séquence d'opérations est-elle parallélisable? Expliquez. (0,5 point)

   Oui, car on peut effectuer op1 et op2 en parallèle, et op3 et op4 en parallèle (entre autres).

3. On considère trois instructions suivantes:

   - `create(instructions)` permet de créer une thread à partir d'une liste d'instructions. Le résultat de cette fonction est un identifiant de thread;
   - `join(id)` permet d'attendre la fin de la thread *id* avant de continuer.
   - `print(var)` permet d'afficher la valeur de la variable *var*.

   En utilisant uniquement les 5 opérations ci-dessus et les deux instructions proposées pour les thread, écrivez un programme en pseudo-code qui effectue les même calculs tout en parallélisant tout ce qui peut l'être puis affiche la valeur de la variable e. (1,5 point)

```
t1=create( op1 )
t2=create( op2 )
t3=create( join(t1) ; op3 )
t4=create( join(t1) ; join(t2) ; op4 )
join(t3)
join(t4)
op5
print(e)
```

   On peut imaginer qu'une thread t5 est créée pour les 3 avant-dernières instructions, à condition d'avoir un join(t5) dans le programme principal.