Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Filière : Télécommunications

Systèmes d'exploitation : examen écrit **Travail écrit du 12.1.2018**

Classe	T-2adfg	Nom de l'étudiant-e	Zambon Yannick		
Enseignant-e	Jacques Supcik				
Durée	95 min.	Points obtenus	50 /50	Note	6.0

BRAVO !

Documentation admise : ***Aucune.***

Directives / Indications

1. Lisez bien chaque question **jusqu'à la fin** avant de commencer à répondre.
2. Répondez aux questions dans les cases prévues à cet effet.
3. La taille des cases ne correspond pas forcément à la taille des réponses. Certaines cases seront donc beaucoup plus grandes que la réponse attendue.
4. Répondez à toutes les questions. Même si vous n'êtes pas sûr. N'importe quelle réponse à plus de chance de marquer des points que pas de réponse du tout.
5. Si vous n'arrivez pas à calculer le résultat numérique pour une question, donnez la formule mathématique qui permet d'obtenir le résultat.
6. Si vous avez besoin de plus de place, utilisez les pages blanches à la fin de ce document. Indiquez alors clairement le numéro de la question.
7. Si vous avez besoin d'encore plus de place, vous faites quelque chose de faux.
8. Vous pouvez écrire au crayon à papier ou au stylo effaçable (Pilot FriXion).
9. Écrivez lisiblement. Toute réponse illisible sera considérée comme fausse.
10. Si vous n'êtes pas sûr de saisir le sens d'une question, expliquez comment vous l'avez comprise.
11. Si la question est ambigüe ou si elle permet plusieurs interprétations, citez les différentes interprétations possibles et développez l'interprétation qui vous semble la plus probable.
12. Si vous ne vous souvenez plus de quelque chose, expliquez le *raisonnement* qui permettrait de trouver la solution.
13. Comme mentionné dans l'article 29 du «Règlement sur la formation de base en HES-SO», toute tricherie ou tentative de tricherie sera pénalisée par une note de 1 **pour l'entier du cours** et la non-acquisition des crédits ECTS correspondants.

Barème

Question	Points	Score
1	10	9
2	10	10
3	10	8
4	10	10
5	10	10
6	0	3
Total :	50	50

1. (10 points) (a) Sur une machine GNU/Linux, on trouve les fichiers suivants :

```
/dev/random
/dev/zero
/dev/snd/timer
```

Décrivez, avec le plus de détails possible, tout ce que vous pouvez dire à propos de ces fichiers

Ces fichiers sont des fichiers spéciaux de type "caractères"

/dev/random permet d'obtenir en lecture une valeur aléatoire.

/dev/zero contient uniquement des 0 et permet donc d'en obtenir également en lecture

/dev/snd/timer contient l'heure ~~actuelle~~ (ou du moins, celle de l'OS)

2

(c) Quelles sont les principales différences entre en système de fichier FAT et un système basé sur des i-nodes ? Quels sont les avantages et inconvénients des deux systèmes ?

La principale différence entre un système FAT et un système basé sur les i-nodes est que FAT garde en RAM tous les blocs des fichiers tandis qu'avec les i-nodes, seuls les fichiers "ouverts" sont en mémoire. ? pas les blocs, mais les références !

En outre, dans un système FAT, le nom des fichiers est stocké avec ses attributs dans un repertoire tandis que les i-nodes possèdent les attributs du/des ~~noms de~~ fichier lié à ceux-ci. ???

Pour ces raisons un système FAT ne peut avoir de hard-link.

Le système FAT est relativement rapide et nécessite peu de "lecture" ou "écriture" pour accéder à un bloc donné, mais il est rapidement pénalisé pour les ~~gros~~ disques à grande capacité, la mémoire étant vite saturée. OK

2

Les systèmes utilisants les i-nodes ne sont, au contraire, pas sujet à ce problème mais nécessite beaucoup de "lecture" pour accéder à un bloc désiré. OK

- pas de hardlink en FAT

(b) Quand on affiche le contenu d'un répertoire (ou dossier), on voit les éléments suivants (un point, deux points disposés horizontalement) :

```
.
..
```

Que représentent ces éléments ? Est-ce qu'ils sont dans tous les répertoires ou y a-t-il des exceptions ?

- . représente le répertoire courant.
- .. représente le répertoire parent.

Ces éléments sont présents dans presque tous les répertoires :

- .. ne peut pas être dans le root directory, ce dernier n'ayant pas de parent.
- . quand à lui est présent partout.

2

(d) Pourquoi est-ce que les SSD sont beaucoup plus rapides que les disques durs magnétiques ? Et si ils sont si rapides, pourquoi utilise-t-on encore des disques magnétiques ? Est-ce que cette différence de vitesse entre les SSD et les disques magnétiques va diminuer ces prochaines années ?

Les SSD n'ont pas de "tête" de lecture, ils ne sont donc pas concerné par le "seek time" d'un HDD qui est assez nettement le facteur limitant d'un point de vue de la vitesse.

Les différences de vitesse ~~sont~~ risquent de s'aggraver. Les technologies évoluent pour les SSD tandis que le "seek time" a très peu changé au fil des ans et ne va surement pas beaucoup s'améliorer.

Plusieurs facteurs peuvent ~~faire~~ favoriser un HDD ou un SSD, le coût, par exemple.

3

9

2. (10 points) Expliquez, en pseudo-code, comment vous implémenteriez la commande LINUX `cat`. Dans votre pseudo-code, utilisez les appels système correspondants et, pour chaque appel système, expliquez le rôle des paramètres et de la valeur de retour.

   Votre pseudo-code devrait comprendre un ou plusieurs *descripteur(s) de fichier (file descriptor(s))* expliquez ce que c'est, à quoi ils servent, et donnez aussi leur *type* et leur *valeur*.

1 Un descripteur de fichier est une clé abstraite sous la
1 forme d'un entier permettant d'accéder à un fichier. Par défaut,
3 de ces descripteurs sont utilisés : 0 - stdin - pour le flux d'entrées
1 - stdout - pour le flux de sortie standard
2 - stderr - pour le flux de sortie d'erreur.

le processus
↓
Le système assignera toujours la plus petite clé disponible par défaut.

---

1 cat file : f = open (file) = 3 → ouvre le fichier passé en paramètre
1 et renvoie le file descriptor assigné.

1 for line in f : ok

1+1 while line > 0 : → jusqu'à la fin du fichier

1+1 l = read (line 3) ; f SIZE? → lit une ligne (concrètement une taille en octet) du fichier lié au descripteur 3. renvoie le ~~nombre de caractères~~ les caractères lus ~~(ou le nombre de com~~

1 write (l , 1) ; → écrit les caractères contenus dans l dans le fichier lié au descripteur 1 (ici stdout) renvoie le nombre de caractères écrits. (-1 s'il y a rien à écrire)

1 close (3) ; f → ferme le fichier passé en paramètre et rend disponible son descripteur. renvoie -1 si tout se passe bien

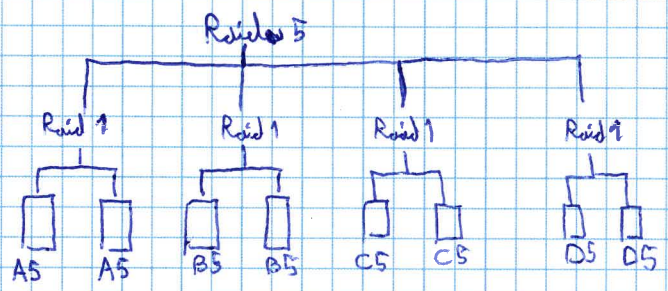
10

3. (10 points) Han Solo a décidé de refaire le système de stockage de son vaisseau, le « Faucon Millenium ». Pour stocker l'ensemble de ses données, il estime avoir besoin de 14 Petabyte (1 Petabyte = 1024 Terabytes = 10^5 Bytes). Comme les passages en vitesse lumière mettent les disques à rude épreuve, Han à besoin d'un système qui soit très robuste qui fonctionne même si il perd plusieurs disques.

Chez Watto, on trouve des disques de 3 Petabytes qui coûtent 12 peggats[1] et des disques de 5 Petabytes qui coûtent 18 peggats.

Proposez deux configurations possibles et expliquez les avantages et les inconvénients de chaque configuration.

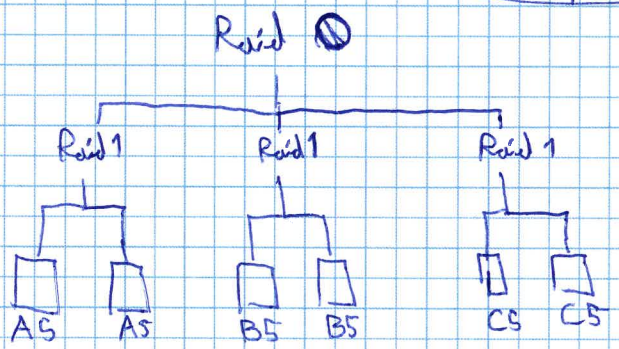
1.



3

- 15 Petabytes d'espace. ✓
- coûte 144 peggats TRÈS CHER !!
- ~~Peut perdre~~ Robuste ! Peut perdre jusqu'à 5 disques (2 sur la même branche, 1 sur les 3 autres). ~~Bonne pour les~~ et supporte au minimum la perte de 3 disques.
- Détection d'erreur et plusieurs accès en lecture en parallèle possible.
  (et pour la modique somme de 36 peggats supplémentaires, on peut opter pour un Raid 6 encore plus robuste)

2.



5

- 15 Petabytes d'espace ✓
- coûte 108 peggats ✓
- Moins robuste : Peut perdre entre 1 et 3 disques. ✓
- Plusieurs accès en lecture en parallèle possible ✓
- Pas de détection d'erreur si les deux copies du disques "se trompent". ?

8

[1] monnaie Hutt, en or, qui est utilisée sur Tatooine

4. (10 points) Luke vous demande de refaire le système de fichiers, basé sur des i-nodes, de son robot R2-D2. R2-D2 doit stocker beaucoup de fichiers, et pour éviter le gaspillage, le système de fichiers utilisera des blocs de 1024 Bytes. Comme les disques sont très grands, les adresses des blocs sont sur **64 bits**.

   Votre système doit permettre d'accéder très rapidement aux premiers blocs d'un fichier et comme certains messages holographiques que R2-D2 doit stocker peuvent être très gros, votre système doit être capable de gérer des fichiers d'au moins 100 GiBytes.

   Dessinez ou décrivez la structure de l'i-node que vous proposez et indiquez la taille maximum de fichiers que votre système permet.

pointeur : 2^3 Bytes. bloc : 2^{10} bytes.

Un bloc de pointeurs contient $2^7 = 128$ pointeurs

Un pointeur direct permet de stocker 1 Ko

Un pointeur indirect permet de stocker 128 Ko

Un pointeur double indirect permet de stocker $(128)^2$ Ko $= 2^{14}$ Ko $= 16$ Mo

Un pointeur triple indirect permet de stocker $(128)^3$ Ko $= 2^{21}$ Ko $= 2$ Go

Un pointeur quadruple(!) indirect permet de stocker $(128)^4$ Ko $= 2^{28}$ Ko $= 256$ Go

Cela fait un peu beaucoup, mais les nouvelles technologies holographiques en développement peuvent pousser plus loin l'espace nécessaire pour 1 fichier.

Pour pouvoir supporter efficacement des tailles de fichiers différentes, on propose à Luke un système contenant : 10 pointeurs direct
2 pointeurs indirect
1 pointeur indirect double
1 pointeur indirect triple
1 pointeur indirect quadruple.

Pour taille supportée totale de 258 Go + des poussières (16 Mo + 266 Ko)

10

5. (10 points) (a) Qu'affiche le programme suivant ?

```
def printSomething(n):
    print("\n".join([" "*(n-i-1)+"*"*(i*2+1) for i in range(n)]))

printSomething(4)
```

A peu près ça :

```
   *
  * * *
 * * * * *
* * * * * * *
```

La fonction écrit à chaque nouvelle ligne ("\n".join) les éléments du tableau en paramètre.
Ce dernier contient des strings avec des espaces répétés (" "*...) (n-i-1) fois suivi d'étoiles répétées (i*2+1) fois.
n étant le paramètre et i l'incrément de la boucle for intégrée, allant de 0 (1e élément du tableau) à n-1 (dernier élément du tableau).

5

Ce qui nous fait donc une belle pyramide de hauteur n, centrée, et de base (n-1)2 + 1.

(b) Décrivez l'expression régulière ci-dessous. A quoi peut-elle servir ? Donnez des exemples de textes que cette expression régulière laisse passer.

```
<([\w]+).*>(.*?)<\/\1>
```

[\w]+ : ~~mot~~ au moins un mot

(.*?) : 0 ou plusieurs caractères

Ce regex permet de trouver des paires de balises Html, par exemple, quelque soit les attributs de la balise et les caractères entre la paire.

< p1 id="# titre"> La recherche du travail écrit d'OS </p1> serait trouvé.

<([\w]+)> : < p1 >

<([\w]+).*> : < p1 blablabla >

(.*?) : n'importe quoi (sauf des sauts à la ligne)

<\/\1> : <\/([\w]+)> : </ p1>.

<Html></Html> serait donc aussi trouvé alors que <john></bob> non : la balise doit être la même.

5

10

Question Bonus

6. (5 points (bonus)) Une machine a des adresses virtuelles sur 48 bits et des adresses physiques sur 32 bits.

   (a) Si les pages font 4 KiB, combien d'entrées y a-t-il dans la table des pages pour une pagination à un niveau ? Expliquez.

4 Ko ⇒ 2^{12} bits d'offset → 48 − 12 = 2^{36} pages virtuelles.

il y a donc 2^{36} entrées dans la table de page. Avec une pagination à un niveau, 1 page = 1 entrée.

3

   (b) Dans le contexte de la mémoire virtuelle, qu'est-ce que le *TLB*, à quoi sert-il et comment fonctionne-t-il ?

0

3