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

Filière : Télécommunications

Systèmes d'exploitation : examen écrit **du 24.11.2017**

Classe	T-2adfg	Nom de l'étudiant-e	Zambon Yannick		
Enseignant-e	Jacques Supcik				
Durée	95 min.	Points obtenus	55 /60	Note	6.0

Bravo !

*Documentation admise : **Aucune.***

Indications

- Lisez bien chaque question **jusqu'à la fin** avant de commencer à répondre.
- Répondez aux questions dans les cases prévues à cet effet.
- La taille des cases ne correspond pas forcément à la taille des réponses. Certaines cases seront donc beaucoup plus grandes que la réponse attendue.
- 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.
- Si vous n'arrivez pas à calculer le résultat numérique pour une question, donnez la formule mathématique qui permet d'obtenir le résultat.
- 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.
- Si vous avez besoin d'encore plus de place, vous faites quelque chose de faux.
- Vous pouvez écrire au crayon à papier ou au stylo effaçable (Pilot FriXion).
- Écrivez lisiblement. Toute réponse illisible sera considérée comme fausse.
- 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	9
4	10	7
5	10	10
6	10	10
Total :	60	55

1. (10 points) Dans le cours, nous avons étudié plusieurs algorithmes d'ordonnancement des processus.

   Dans un exercice, nous avions 5 jobs de traitement par lots (batch), de A à E, qui arrivaient dans un centre informatique pratiquement au même instant. Leurs délais d'exécution respectifs étaient de 10, 6, 2, 4 et 8 minutes. En comparant les algorithmes d'ordonnancement, on avait trouvé que l'algorithme «le job le plus court en premier» était préférable aux autres.

   Selon quel(s) critère(s) avons-nous pu tirer cette conclusion ?

On part ici du principe que l'on a qu'un seul processeur pouvant ne faire tourner qu'un seul processus à la fois.

On accepte aussi le fait que ~~l'exécution~~ le temps d'exécution était connu à l'avance et que les process sont tous indépendants.

On remarque alors que "le plus court en premier" est préférable car le moyenne des temps d'attente depuis leur arrivée est la plus petite possible.

0h

Dans le cas présent on aurait : $\frac{2+6+12+20+30}{5} = 12$ min.

Les autres "ordres" donnent un temps moyen plus grand. Ici le critère est donc d'avoir un maximum de "job/heure"

délai de rotation

1/2

Citez d'autres critères qui permettent de comparer les algorithmes d'ordonnancement.

On peut également considérer le fait de partager du mieux les ressources du CPU ok
ou d'octroyer un quantum fixe assez long pour obtenir une meilleure performance.

On peut aussi mettre l'accent sur la priorité d'un processus par rapport à d'autres ou simplement favoriser l'ordre d'arrivée. 1/2

1/2

- Equité
- Politique appliquée
- Equilibre
- Délai de Rotation
- Utilisation CPU
- temps de réponse
- Proportionnalité

Expliquez ce qu'est un ordonnanceur **préemptif**.

C'est un ordonnanceur capable d'arrêter un processus quand il estime que c'est nécessaire.

2

Quel est le contraire de préemptif (ou comment nomme-t-on un algorithme non-préemptif) ?

J'aurais bien répondu "non-préemptif"... Coopératif !

1/2

Les algorithmes enseignés par Lots sont non-préemptifs.

Quelles sont les conditions pour qu'un système non-préemptif fonctionne correctement ?

Il faut une coopération des processus pour que l'utilisation du CPU soit maximale.

Comment ? → en cédant la main régulièrement.

1

L'algorithme de vieillissement (avec $a = 1/2$) est utilisé pour estimer des délais d'exécution des processus. Les quatre premières exécutions, de la plus ancienne a la plus récente, ont été de 30, 40, 45 et 50 µs. Quelle va être l'estimation du prochain délai d'exécution ?

1. $\frac{30}{2} + \frac{40}{2} = 35\,\mu s$

2. $\frac{35}{2} + \frac{45}{2} = 40\,\mu s$

3. $\frac{40}{2} + \frac{50}{2} = 45\,\mu s$

2

9

2. (10 points) Les processus que nous observons sur une machine avec un processeur passent 60% de leur temps dans le CPU et 40% dans les I/O.

   (a) A partir de combien de processus le taux d'utilisation du CPU dépassera les 70% ?

taux utilisation CPU = $1 - p^n$ ✓ où n = le nombre de processus et p = le % dans les I/O

1

On veut donc que $1 - 0{,}4^n > 70\%$

ce qui est déjà le cas pour $n=2$ ✓

4

   (b) A partir de combien de processus le taux d'utilisation du CPU dépassera les 80% ?

On veut que $1 - 0{,}4^n > 0{,}8$

ce qui est à nouveau le cas pour $n=2$ ✓

2

(c) A partir de combien de processus le taux d'utilisation du CPU dépassera les 90% ?

On cherche à avoir $1-0,4^n > 0,9$

On obtient ça dès $n=3$

OK... mais comment ?

2

(d) Combien de processus faut-il pour atteindre un taux d'utilisation du CPU de 100% ?

Ici il faudrait que $0,4^n = 0$

Pour toutes les valeurs entre 0 et 1, c'est le cas quand n tend vers l'infini.

Il faudrait donc théoriquement une infinité de processus.

2

10

3. (10 points) (a) Expliquez comment sont structurés (organisés) les processus dans un système Linux (liste, arbre, graphe acyclique, graphe orienté acyclique, ...) Citez les principaux attributs des processus (champs d'une entrée de la table des processus).

Dans un système Linux, les processus sont organisés en arbres.
Un processus est toujours rattaché à un processus-parent (sauf la racine).
Si un processus se termine avant ses enfants, ces derniers sont automatiquement rattachés à la racine. Pour créer un "nouvel enfant", on duplique le processus parent (fork()) et on modifie son contenu.

Un processus contient de nombreux attributs, on peut citer:

- Son PID
- Ses pointeurs vers les segments (text, data, ...)
- Sa stack
- Ses propriétés de "fichiers" → USER Id, Group Id, ...
- Sa priorité.

4

(b) Expliquez ce qui se passe lorsqu'un nouveau processus est créé, et expliquez où ce nouveau processus se situe dans la structure que vous avez mentionnée au point précédent. Indiquez aussi les appels système qui sont liés à la création de processus.

J'ai apparemment déjà répondu, malheur!

→ Lorsque un nouvel enfant est créé, il se place sous le processus dont il a été dupliqué par l'appel système fork. On peut alors modifier son "contenu" à l'aide d'autres appels systèmes (~~[illegible]~~ ?)

exec
aussi

3

(c) Expliquez ce qui se passe lorsqu'un processus termine (de son plein gré ou non). Expliquez ce qui se passe dans la structure en fonction des différents cas possibles. Indiquez aussi les appels système qui sont liés à la terminaison d'un processus.

Quand un processus se termine, il sort de l'arbre et le fonc est remis par son parent. Si le processus terminé avait encore des enfants, ces derniers sont rattachés à la racine. Pour les appels systèmes, on note ~~execve()~~ ?

exit, wait pid, Kill

2

9

4. (10 points) Vous devez concevoir un MMU pour une mémoire virtuelle. On vous demande de faire un système avec des pages de 8 KiB et des tables de pages qui tiennent chacune sur une seule page. On souhaite aussi un système avec 3 niveaux de tables de pages.

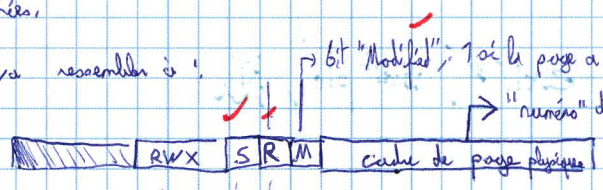
   On aimerait avoir des entrées de table de pages de 4 Bytes. En plus de 3 bits de permissions, on aura les 3 bits nécessaires pour une MMU fonctionnelle. Décrivez une entrée d'une table de pages (contenu, structure, …).

Si on veut des pages de 8 KiB (2^{13}) et des entrées de 4 bytes (2^2) :

Pour que la table de page tienne sur une page, il faut qu'elle contienne $\frac{2^{13}}{2^2} = 2^{11}$ entrée

⇝ 2k entrées.

Une entrée va ressembler à :



↳ bit "Modified", 1 si la page a été modifiée.

→ "numéro" de la page physique référencée. Le nb de bits dépend du nombre de pages physiques.

32 − 1 = 21

↳ 3 bits de permission écriture, lecture, exécution

↳ bit de statut, 1 si l'entrée référence une page physique

↳ bit "Referenced", 1 si la page a été lue, périodiquement remis à 0

1

Combien aurez-vous de table(s) de premier niveau et combien d'entrées y aura-t-il dans cette/ces table(s) ?

Comme calculé au point précédent, une table de premier niveau contient 2^{11} entrées de 4 bytes pour tenir sur une seule page de 8 KiB.

Il y a une seule table de premier niveau, celle-ci "pointera" sur toutes les tables de deuxième niveau (jusqu'à ~2000, donc).

2

Est-ce que la/les table(s) du deuxième niveau auront le même nombre d'entrée(s) que la/les table(s) de troisième niveau ? Expliquez votre réponse.

Pas forcément, cela dépend du nombre de bit à disposition pour l'adressage et la ~~nombre de~~ taille d'une page, mais dans l'idéal, si l'on veut que chaque table de page aille exactement dans une seule page, on souhaite que les 3 niveaux contiennent le même nombre d'entrée : 2^{11}.

Cependant, en utilisant déjà 11 bits pour le premier niveau et 13 bits pour l'offset, il faut que $11 + 13 + n$ bit 2e niveau $+ n$ bit 3e niveau $= 2^k$

Pour ~~une adr~~ un espace d'adressage de 32 bits, il ne reste que 8 bits à disposition des deux niveaux.

1

Quel sera l'espace d'adressage de la **mémoire virtuelle** ? Expliquez votre réponse.

L'espace d'adressage de la mémoire physique est la somme des bits nécessaires à chaque niveau + les 13 bits d'offsets de la page physique.

OK

2

~~a 11 bits par niveaux~~

11	11	11	13

$2^{46} = 64$ To

Quelle sera la taille maximum de la **mémoire physique** que pourra avoir votre système ? Expliquez votre réponse.

La taille de la mémoire physique n'est pas vraiment liée à la mémoire virtuelle, mais dans l'idéal, on souhaite que la mémoire physique soit au maximum de la même taille que la mémoire virtuelle pour que chaque page puisse être référencé.

?

1

cadre	offset
26	13

$\rightarrow 2^{39} = 512$ Go

7

5. (10 points) (a) Ecrivez une expression régulière permettant de reconnaître une heure au format 24h. On ne considère que les heures et les minutes (pas de seconde), et on sépare les heures des minutes avec soit le caractère «:», soit avec la lettre «h». Les heures suivantes sont toutes valides :

- 8:45
- 09h30
- 23:59
- 0h00

~~/[[:digit:]]{1,2} (:|h) [[:digit:]]{2} /~~

/ ([0-1]? [0-9] | 2 [0-3]) (:|h) [0-5] [0-9] /

(b) Ecrivez une nouvelle expression régulière permettant de reconnaître une heure au format «12h»[1]. Dans ce format, l'heure ne dépasse pas 12 et elle est suivie de «a.m.»[2] pour le matin et «p.m.»[3] pour l'après-midi. La table ci-dessous donne l'équivalence entre le format 24h et le format 12h :

00h00 (minuit)	12:00 a.m.
00h01	12:01 a.m.
01h00	1:00 a.m. (ou 01:00 a.m.)
11h00	11:00 a.m.
11h59	11:59 a.m.
12h00 (midi)	12:00 p.m.
12h01	12:01 p.m.
13h00	1:00 p.m. (ou 01:00 p.m.)
23h00	11:00 p.m.
23h59	11:59 p.m.

/ (1 [0-2] | 0?[1-9]) : [0-5][0-9] \s (a\.m\. | p\.m\.) /

10 01 00-59
11 02
12 03

10

[1]Le format «12h» est principalement utilisé aux Etats-Unis
[2]ante meridiem
[3]post meridiem

6. (10 points) Dans un TP, nous avons utilisé la commande «wc» pour compter le nombre de lignes, de mots et de caractères dans un fichier.

   Comment savoir dans quel répertoire se trouve cette commande ?

En utilisant "locate wc" on peut trouver toutes les instances de wc.
Le répertoire de wc sera également l'un de ceux contenu dans la variable $PATH.

04

2

whereis / which

Supposons que vous avez écrit un programme qui fait la même chose que la commande «wc», mais en mieux (plus rapide, plus joli, …)

Vous aimeriez aussi appeler cette commande «wc» et vous aimeriez garder la commande du système au cas où.

comment faire pour que lorsque vous tapez «wc» dans un shell, ça soit votre commande qui soit appelée et non la commande du système ?

En premier lieu il faut s'assurer que wc appelle bien notre commande en nommant notre script "wc" ou en utilisant un alias, OK

2

Il faut ensuite s'assurer que ~~l~~ notre wc se situe dans un des répertoires de $PATH. où ?

C'est l'ordre dans le $PATH qui compte

Pour le reste, j'ai oublié !

Supposons que nous sommes connectés à une machine Linux et que nous tapions les commandes suivantes :

```
$ mkdir folder1
$ cd folder1
$ echo "Hello" > file1
$ ls
```

Que voit-on à l'écran après l'exécution de ces commandes ? (On suppose que le répertoire forlder1 n'existait pas encore avant la première commande)

Une liste des fichiers du ~~dossier~~ répertoire courant ; en l'occurence uniquement :

file 1

2

Tapons maintenant les commandes suivantes :

```
$ ls >list1
$ cat list1
```

Que voit-on alors à l'écran ? Expliquez le raisonnement qui vous à permis de donner cette réponse.

file 1
list 1.

A Je considère ici que list1 est créé avant l'exécution de la commande ls.
"file1
list1" sera alors stocké dans le fichier list1. cat list1 permet d'afficher son contenu dans le terminal.

4

10