

Pale INF422 – Composants d'un système informatique

INF422 Final Exam – Components of a Computing System

2009–2010

- L'examen dure 3 heures.
The exam lasts 3 hours.
- Tous documents autorisés.
All documents are authorized.
- Il est impératif de commenter les programmes et de justifier vos réponses.
Programs must be commented and every answer must be justified.

Pale supplémentaire

1 Questions diverses

Il s'agit d'un questionnaire à choix multiple. Écrivez sur votre copie le numéro de la réponse choisie.

Notation : **1** point par bonne réponse, **0** points en l'absence de réponse, **–0,5** points par réponse fausse. La note obtenue entre **–2** et **4** est ensuite ramenée au nombre de points accordé à l'exercice dans le barème.

Question 1.1

Laquelle de ces commandes permet d'afficher le contenu du fichier `fic` une page à la fois ?

- (1) `man fic` (2) `more > fic` (3) `cat fic | more` (4) `man fic > more`

Question 1.2

À quoi sert la variable `PATH` ?

- (1) à chercher des fichiers texte ;
(2) à chercher des fichiers exécutables ;
(3) à spécifier le répertoire d'accueil de l'utilisateur ;
(4) à chercher des fichiers de données.

Question 1.3

La ligne suivante est affichée par la commande `ls -al` :

```
drwxr-x--- 2 titi admin 4096 Aug 31 15:50 tata
```

Que décrit elle ?

- (1) un répertoire `tata` lisible, inscriptible et exécutable pour l'utilisateur `titi` sont `rw` ;
(2) un répertoire `titi` lisible et exécutable pour le groupe `admin` ;
(3) un fichier régulier `titi` de taille 2048 ;
(4) un répertoire `tata` contenant 2048 fichiers et sous-répertoires.

Question 1.4

Qu'est-ce que la pagination mémoire (*paging*) ?

- (1) un mécanisme matériel permettant de séparer le code des données des processus ;
(2) un mécanisme logiciel permettant de communiquer avec les périphériques ;
(3) un mécanisme matériel pour la traduction d'adresse et la protection mémoire ;
(4) un mécanisme logiciel pour afficher des textes page par page.

2 Script mystère

Question 2.1

Commentez le script shell suivant, et décrivez sa fonction et son fonctionnement lorsque l'on l'exécute avec l'argument (en ligne de commande) `ls`.

```
#!/bin/sh

for program in "$@"; do
  list='echo $PATH | sed -e 's:// /g''
  file=$list/$program
  [ -x "$file" ] && echo $file
done
```

Question 2.2

Que se passe-t-il dans le cas où `list` contient plusieurs mots séparés par des espaces ?

Question 2.3

Modifiez le script pour traiter ce cas correctement.

Question 2.4

Que se passe-t-il désormais lorsque l'on exécute le script corrigé avec l'argument `x*`.

3 Communication et signaux

On souhaite réaliser une commande `compte` qui s'exécute sous forme de deux processus P_1 (le père) et P_2 (le fils). Les deux processus sont programmés en shell ; vous pourrez utiliser des fichiers distincts pour les scripts implémentant ces processus. Ils communiquent par un socket IP par l'intermédiaire de la commande `nc`.

On souhaite que P_1 et P_2 réalisent le processus itératif suivant :

- P_2 lit une ligne de l'entrée standard et envoie celle-ci dans le socket ;
- P_1 lit une ligne dans le socket (il reste bloqué tant qu'il n'y en a pas) ;
- P_1 attend une seconde puis affiche la ligne lue ;
- Lorsque 100 lignes ont été lues P_1 envoie le signal `SIGINT` (le signal numéro 2) à P_2 pour l'interrompre, puis P_1 termine.

Question 3.1

Implémentez le programme `compte`.

Question 3.2

Que se passe-t-il si P_2 atteint la fin de son entrée standard avant d'avoir lu 100 lignes ? Expliquez le comportement du processus en fonction du fichier (régulier ou non) associé à cette entrée standard.

4 Codage des fichiers attachés

Un des moyens disponibles pour envoyer des fichiers binaires (images, sons, etc.) dans des courriers électroniques consiste à les *encoder*, c'est à dire à les transformer de manière à ce qu'ils ne contiennent que des caractères imprimables. Bien entendu, cette transformation doit être bijective, le destinataire devant pouvoir *décoder* le fichier lors de la réception.

L'encodage base64, défini par la norme Internet MIME, consiste à lire les octets 3 par 3, formant ainsi des groupes de 24 bits, puis à les regrouper en 4 blocs de 6 bits. Chaque bloc a donc une valeur comprise entre 0 et $2^6-1 = 63$, qui est convertie en un caractère suivant le tableau :

valeur	caractère	valeur	caractère	valeur	caractère	valeur	caractère
0	A	26	a	52	0	62	+
1	B	27	b	53	1	63	/
2	C	28	c	54	2		
			...				
25	Z	51	z	61	9		

Ainsi, la séquence de 3 octets de valeur 0x41, 0x62, 0x63 s'écrit en binaire : 0100 0001, 0110 0010, et 0110 0011, ce qui donne donc les 4 valeurs 010000, 010110, 001001, 100011, soit 16, 22, 9 et 35. Ces valeurs sont converties en la chaîne QWJj.

Dans le cas où la taille du fichier binaire ne serait pas un nombre multiple de 3, les un ou deux octets manquants sont remplacés par des octets nuls pour réaliser la transformation. Une fois les 4 caractères obtenus, les deux derniers caractères (s'il manque deux octets) ou le dernier caractère (s'il manque un octet) sont remplacés par le symbole =.

Question 4.1

À quoi sert la méthode Java suivante ? Vous indiquerez quelles lignes de code impliquent un service effectué par le noyau du système d'exploitation.

```
static byte[] whatisit(String name) {
    byte[] buffer;
    try {
        File file = new File(name);
        int size = file.length();
        buffer = new byte[size];
        FileInputStream in = new FileInputStream(file);
        while (size > 0) {
            n = in.read(buffer, position, size);
            position = position + n;
            size = size - n;
        }
        in.close();
    } catch (Exception e) {
        System.exit(1);
    }
    return buffer;
}
```

Question 4.2

Programmez une méthode Java `encoder()` qui prend sur l'entrée standard (`System.in`) un fichier binaire quelconque et produit sur la sortie standard (`System.out`) la transformation en base64.

Question 4.3

Programmez une méthode Java `decoder()` qui prend sur l'entrée standard un fichier base64, et restitue sur la sortie standard le fichier binaire original.

Question 4.4

Afin d'améliorer les performances de ces fonctions, on souhaite les paralléliser en utilisant plusieurs threads de calcul. On s'intéresse par exemple au décodeur. Le thread principal est dédié à la lecture de l'entrée standard en base64 et à l'écriture sur la sortie standard en binaire ; un ensemble de n threads ($n \geq 1$) effectue la traduction. Décrivez le plus exhaustivement possible, mais sans écrire de code Java, les modifications à apporter à la méthode de décodage pour la paralléliser.