Introduction aux systèmes d'exploitation (IS1)

Examen du 16 Janvier 2007

Durée: 2 heures

- Les notes manuscrites ainsi que les énoncés de TP sont autorisés.
- **Les livres sont interdits.**
- Le barème est donné à titre indicatif et est susceptible d'être modifié.
- Un soin particulier devra être apporté à la présentation ainsi qu'à l'orthographe.
- La note prendra compte de la clarté des explications et du raisonnement.
- Ce sujet fait 4 pages (plus annexe).

Exercice 1 – *Questions rapides - [10 points]*

Cet exercice comporte des questions courtes auxquelles vous devez vous efforcer de répondre de façon **justifiée** mais **concise** (deux ou trois lignes).

Il suffira de répondre correctement à 20 questions pour obtenir la totalité des points. Aucun bonus n'est attribué en cas de dépassement.

Système de fichiers.

1. Si l'on se trouve dans le répertoire `/usr/local/firefox/`, quel est le nom relatif du fichier `/usr/bin/xclock` ?
2. Si l'on se trouve dans le répertoire personnel de l'utilisateur `dupond`, lui-même situé dans le répertoire `/home/`, quel est le nom absolu le plus court possible du fichier `if1/../../durand/./dugenou/is1/../liste_courses.txt` ?
3. Si l'on se trouve dans le répertoire `/tmp/`, quelle commande taper pour créer un sous-répertoire `is1` dans son répertoire personnel sans changer de répertoire courant ?
4. Toujours depuis `/tmp`, quelle commande taper pour recopier le fichier `rapport.txt` sous le nom `compte-rendu.txt` dans le répertoire créé à la question précédente ?
5. Comment peut-on effacer le répertoire `/tmp/a-effacer/` si l'on suppose que ce répertoire n'est pas vide (et que l'on dispose des droits suffisants) ?

Droits. On donne le listing suivant :

```
$ ls -lR
---xr--r--  1 michel  etudiants   12  5 Jan 18:06 fic.sh
drwxr--r-x  2 michel  etudiants  102  5 Jan 18:06 rep

./rep:
-rw-r-----  1 michel  etudiants  306  5 Jan 18:06 fic.txt
drw-----w-  1 michel  etudiants  306  5 Jan 18:06 sous-rep
```

6. La ligne concernant le répertoire `rep` contient un certain nombre d'informations. En particulier, le chiffre `2` désigne le nombre de liens physiques sur ce répertoire, et `102` est sa taille sur le disque. A quoi correspondent les autres informations ?
7. Quels utilisateurs peuvent lire le contenu du fichier `fic.txt` ?
8. Quels utilisateurs peuvent afficher le contenu du répertoire `rep` ?
9. Quels utilisateurs peuvent créer un nouveau fichier dans le répertoire `sous-rep` ?
10. Quels utilisateurs peuvent exécuter le script `fic.sh` ?

Processus.

11. Que désigne l'abréviation « PID » ? Expliquez ce concept en quelques mots.
12. Quelles sont les différences entre les signaux `SIGSTOP`, `SIGKILL` et `SIGTERM` ?
13. Expliquez à quoi sert la variable « `?` » et en quoi elle intervient dans une construction `if` ou `while`.
14. Écrivez une commande qui compile le fichier java `MaClasse.java`, puis l'exécute **uniquement** si la compilation n'a pas produit d'erreurs (2 réponses possibles).

Redirections.

15. Dans quelles circonstances la commande `ls fichier 2> /dev/null` peut-elle ne rien afficher ?
16. Expliquez le concept de processus filtre.
17. Expliquez le concept de tube (*pipe*) et à quoi il sert.
18. Que fait la commande `grep "public" *.java | tail > res.txt` ? Écrivez une commande composée équivalente n'utilisant pas de tube.

Tests.

19. Que teste la commande suivante :
    `[ -r Main.java -a ( ! -e Main.class -o ( Main.java -nt Main.class ) ) ]`
20. Quelle variable d'environnement a pour valeur le chemin du répertoire personnel de l'utilisateur ? Écrire une commande qui teste si le répertoire courant est le répertoire personnel et si ce répertoire est accessible en traversée.

Boucles.

21. En utilisant des tests appropriés, écrivez une boucle qui ajoute les préfixes suivants au nom de chaque fichier du répertoire courant selon sa nature :
    – `L_` pour les liens symboliques.
    – `D_` pour les répertoires.
    – `F_` pour les autres fichiers.

Expressions régulières. Les questions suivantes portent sur les expressions régulières telles que celles qu'utilise la commande `grep`.

22. Quels mots l'expression régulière `b[aei]bo(bu)*` désigne-t-elle ?

23. Quels mots l'expression régulière `(po+m)+` désigne-t-elle ?

24. En Java, il est de bon usage que les noms de variables commencent par une lettre minuscule, suivie d'une séquence quelconque de lettres minuscules ou majuscules et de chiffres. Écrivez une expression régulière qui représente les noms de variables de cette forme.

25. Il est cependant autorisé par le langage (mais découragé par l'usage) d'utiliser les caractères `_` ou `$` dans un nom de variable, même comme première lettre. Écrivez une expression régulière qui décrit les noms de variables corrects mais non recommandés.

Divers.

26. Comment définir une commande `px` équivalente à `ps -aux` ? Comment faire pour pouvoir l'utiliser même après un redémarrage et depuis n'importe quel terminal ?

27. Sous Unix, quelle est la particularité d'un fichier dont le nom commence par un point (`.`) ? Quelle est la différence en pratique entre un tel fichier et un fichier ordinaire ?

28. Quelle ligne de commande taperiez vous pour afficher le texte suivant : `J'ai horreur des caractères spéciaux ( à part * ) ! ! Et vous ?`

29. Expliquez en quelques mots la différence de rôle entre le programme `bash` (shell) et le programme `xterm` (terminal).

Exercice 2 – *Processus [4 points]*

Le but de cet exercice est d'écrire un script de contrôle des processus qui évite à l'utilisatrice d'avoir à manipuler elle-même le PID. Référez-vous à l'annexe jointe pour un exemple d'affichage produit par la commande `ps`.

1. Ecrivez un script `monps.sh` qui affiche un message disant si le nom du processus passé en argument existe. Par exemple, la commande `$ ./monps.sh bash` doit afficher `bash est lancé.`

2. Il est possible que plusieurs processus portant le même nom soient lancés au même moment. Améliorez votre script pour qu'il affiche les PID de tous les processus lancés portant le nom passé en argument.

Par exemple, si `xclock` a été lancé deux fois avec les PID 24 et 30, la commande `$ ./monps.sh xclock` doit afficher `xclock est lancé (2 instances : 24, 30).`

3. Ajoutez à votre script un paramètre **facultatif** permettant de stopper (`-stop`) ou de tuer (`-kill`) le ou les processus en question. Par exemple `$ monps.sh -kill xclock` envoie le signal `SIGKILL` à tous les processus `xclock`. Le script doit afficher un compte-rendu des processus tués ou arrêtés, ou un message d'erreur si les options ne sont pas correctes.

4. On ne souhaite à présent tuer qu'un seul processus portant un nom donné (par exemple celui qui possède le plus grand PID). Pouvez-vous proposer une solution (sans donner le code exact) ?

Exercice 3 – *Problème : carnet d'adresses [6 points]*

Le but de ce problème est de créer un carnet d'adresse artisanal permettant de stocker le prénom, le nom et l'adresse électronique de vos contacts.

Votre carnet d'adresse sera constitué d'un répertoire `.adresses` situé dans votre répertoire personnel. Les informations de chaque contact seront stockées dans un fichier séparé de la forme `nom_prenom.contact` (nom et prénom en minuscules).

Par exemple, pour l'utilisateur Jean-Philippe Smet on créera dans le dossier `.adresses` un fichier nommé `jean-philippe_smet.contact` contenant les informations suivantes :

```
NOM=Smet
PRENOM=Jean-Philippe
EMAIL=jps@johnny.com
```

Nous allons écrire un script `adresses.sh` muni de plusieurs fonctions permettant de gérer ce carnet d'adresses. Si vous ne savez pas répondre à certaines questions, vous pouvez laisser un blanc dans votre programme avec un commentaire expliquant la fonctionnalité manquante.

1. Commencez par écrire la base du script `adresses.sh`. Sans options, le comportement par défaut est d'indiquer combien de contacts sont actuellement enregistrés. Si le répertoire `.adresses` n'existe pas, il faut le créer (en vérifiant que c'est possible). En cas d'échec d'une de ces opérations, affichez un message d'erreur approprié.
2. Nous voulons maintenant pouvoir rajouter des contacts au carnet d'adresse. Pour cela on souhaite pouvoir taper (par exemple) la commande suivante :
   `adresses.sh -ajout Jean-Philippe Smet jps@johnny.com`
   Modifiez votre script en conséquence.
3. L'intérêt d'un carnet d'adresses est de pouvoir y retrouver des informations facilement. Pour cela, ajoutez deux nouvelles possibilités à votre script : une option `-liste` qui affiche tous les contacts enregistrés (un contact par ligne) et une fonction `-rech` qui affiche tous les contacts dont le nom, le prénom ou l'adresse contiennent la chaîne passée en paramètre. Par exemple :

   ```
   adresses.sh -liste        # affiche tous les contacts
   adresses.sh -rech "Smet"  # affiche Jean-Philippe Smet <jps@johnny.com>
   ```

4. Il est important de pouvoir supprimer des contacts erronés ou qui ne sont plus valables. Ajoutez la possibilité d'appeler votre script avec l'option `-suppr` pour supprimer une entrée. Indiquez quels autres paramètres sont nécessaires, et affichez un message d'erreur pertinent en cas de problème.
5. Votre script fonctionne-t-il toujours correctement si l'on essaye de faire un ajout sur un nom existant déjà ? Proposer (sans la programmer) une solution pour résoudre ce problème.

Annexe

Manuel de la commande `test` (extrait)

```
SYNOPSIS
test [expr]
test -help,-version
DESCRIPTION
```

test renvoie une valeur `0` (vrai) ou `1` (faux) suivant l'évaluation de l'expression conditionnelle expr. Les expressions peuvent être unaires ou binaires. Les expressions unaires sont généralement utilisées pour examiner le statut d'un fichier. Il existe également des opérateurs de chaînes de caractères, et des opérateurs de comparaison numérique.

`-d fichier`
: Vrai si le fichier existe et est un répertoire.

`-e fichier`
: Vrai si le fichier existe.

`-f fichier`
: Vrai si le fichier existe et est un fichier ordinaire.

`-L fichier`
: Vrai si le fichier existe et est un lien symbolique.

`-p fichier`
: Vrai si le fichier existe et est un tube nommé.

`-r fichier`
: Vrai si le fichier existe et est lisible.

`-s fichier`
: Vrai si le fichier existe et a une taille supérieure à zéro.

`-w fichier`
: Vrai si le fichier existe et est accessible en écriture.

`-x fichier`
: Vrai si le fichier existe et est exécutable.

`-O fichier`
: Vrai si le fichier existe et appartient à l'UID effectif de l'appelant.

`-G fichier`
: Vrai si le fichier existe et appartient au GID effectif de l'appelant.

`fichier1 -nt fichier2`
: Vrai si fichier1 est plus récent (d'après les dates de modification) que fichier2.

`fichier1 -ot fichier2`
: Vrai si `fichier1` est plus ancien que `fichier2`

`-z chaîne`
: Vrai si la longueur de la chaîne est nulle.

`-n chaîne`
`chaîne`
: Vrai si la longueur de la chaîne n'est pas nulle.

`chaîne1 = chaîne2`
: Vrai si les deux chaînes sont égales.

`chaîne1 != chaîne2`

Vrai si les deux chaînes sont différentes.

`! expr`

Vrai si `expr` est fausse.

`expr1 -a expr2`

Vrai si `expr1` et `expr2` sont toutes les deux vraies.

`expr1 -o expr2`

Vrai si `expr1` ou `expr2` est vraie.

`arg1 OP arg2`

`OP` est dans la liste `-eq`, `-ne`, `-lt`, `-le`, `-gt`, ou `-ge`. Ces opérateurs arithmétiques renvoient vrai si `arg1` est égal, différent, inférieur, inférieur ou égal, supérieur, ou supérieur ou égal à `arg2`, respectivement. `arg1` et `arg2` doivent être des entiers (positifs, ou négatifs) ou l'expression spéciale `-l chaîne`, qui évalue la longueur de la chaîne.

Expressions régulières (syntaxe de grep)

Voici une liste de symboles utilisables pour grep :

.	un seul caractère quelconque
*	répétition, zéro ou plusieurs fois, du caractère précédent. `ab*` représente l'ensemble { a,ab,abb,abbb,...}
+	répétition (au moins une fois) du caractère précédent `ab+` représente l'ensemble { ab,abb,abbb,...}
?	l'élément précédent est facultatif `ab?c` représente l'ensemble {ac,abc}
^	début de ligne `^a` représente toutes les lignes qui commencent par 'a'
$	fin de ligne `a$` représente toutes les lignes qui finissent par 'a'
`[...]`	liste ou intervalle de caractères recherchés [abcd] représente l'ensemble {a,b,c,d} [a-d] représente l'ensemble {a,b,c,d}
`[^ab]`	listes des caractères interdits

Exemples :

- `grep abc fichier` recherche la chaîne `abc` dans toutes les lignes du fichier. Les lignes trouvées sont envoyées sur la sortie standard.
- `grep " " fichier` recherche les lignes qui contiennent un (et un seul) espace entre 2 mots dans `fichier`.

Commande `ps`

Voici le résultat de l'exécution de la commande `ps` :

```
$ ps -o comm,pid
COMMAND           PID
bash            28090
ps              28112
```