Examen du cours Système de Majeure 2.

Durée 2 heures

Les appendices B et C du cours, éventuellement annotées, sont la seule documentation autorisée.

Dans les solutions, on pourra utiliser les fonctions du module `Misc` *décrites dans l'appendice B.7 en les appelant explicitement avec le préfixe* `Misc`.

On s'efforcera de donner des explications à la fois précises et concises.

Les quatre parties sont indépendantes. Les questions soulignées peuvent être plus difficiles ou plus longues que les autres de la même partie.

Partie 1: Fork - Execv

1) Que fait l'appel système `execv` *file argv* ?

2) Peut-il échouer en raison d'une mauvaise valeur de `argv` (justifier très brièvement) ?

3) Est-ce que le *comportement* des signaux est hérité lors de cet appel système (justifier très brièvement) ?

4) Donner les principales étapes du point de vue du système lors de l'appel système `execv` *file argv* ?

5) Que deviennent, par défaut, les descripteurs ouverts lorsqu'un processus fait `execv` ? Quel est l'intérêt de ce choix ?

6) Quel est, à votre avis, la réponse à la question 3) pour l'appel système `fork()` (justifier brièvement) ?

7) Même question que 5) pour l'appel système `fork`.

□

Partie 2: Mkpath

On rappelle qu'en général les appels systèmes qui accèdent au système de fichiers provoquent une erreur `ENOENT` lorsqu'on leur demande d'accéder à un chemin qui n'existe pas.

1) Écrire une fonction `mkpath` de type `string -> int -> unit` telle que l'appel `mkpath` *path perm* crée le répertoire *path* ainsi que tous les répertoires sur le chemin *path* qui n'existent pas. Tous les répertoires créés le seront avec les permissions *perm*. Le chemin *path* est absolu ou relatif. Si la commande `mkpath` échoue, on ne cherchera pas à retirer les répertoires qui auraient déjà pu être créés. Si un chemin intermédiaire de p existe mais n'est

pas un répertoire, on déclenchera une erreur `ENOTDIR` comme le ferait la commande `mkdir` (et la plupart des commandes qui prennent des chemins en arguments).

2) On suppose que le répertoire `./A` existe avec les droits `0740` (*i.e.* `rwxr-----` en notation symbolique). Décrire une valeur de *perm* (en notation octale ou symbolique, ou en français) avec laquelle le répertoire `A/B` a été créé mais le répertoire `A/B/C` n'a pas pu l'être.

3) Donner une autre raison pour laquelle la création du répertoire `A/B/C` peut échouer alors que celle du répertoire `A/B` a réussi.

4) On souhaiterait que la commande réussisse quand même dans une majorité des cas décrits en 2) comme produisant des erreurs, tout en assurant au final que les répertoires créés auront bien les droits demandés. Donner une variante `mkpath_plus` de `mkpath` réalisant cela.

□

Partie 3: Descripteurs en accès direct

On rappelle que l'appel système `lseek` lève l'exception `ESPIPE` si le descripteur auquel il est appliqué n'est pas en accès direct (*i.e.* ne supporte pas `lseek`).

1) Écrire une fonction `seekable` de type `Unix.file_descr -> bool` qui teste si le descripteur reçu en argument est en accès direct.

2) Écrire une fonction `unlinkf` qui se comporte comme `unlink`, mais ne lève pas d'erreur s'il n'est pas possible d'effectuer l'opération.

3) On vous donne la fonction suivante :

```
0  let rec create_file_descr() =
1    let tmpdir = "/tmp" in
2    let rec open_tmp n =
3      try
4        let name = Filename.concat tmpdir (string_of_int n) in
5        let descr = openfile name [ O_CREAT; O_RDWR; O_EXCL ] 0 in
6        unlinkf name;
7        descr
8      with Unix_error (EEXIST, _, _) ->
9        open_tmp (n+1) in
10   open_tmp (getpid());;
```

Que fait cette fonction ?

4) Y a-t-il quelque chose à faire pour se prémunir contre un risque éventuel, si, après le retour de la fonction `create_file_descr`, un fichier est à nouveau créé avec le même nom que celui qui a servi à créer `descr` ?

5) Est-il pour autant garanti que le descripteur de fichier `create_file_descr` donne un accès privé dans le sens où ce qui est lu ou écrit dans `desc` n'a pu être écrit ou lu que par le processus courant ou un de ses descendants ?

6) On suppose que le programme Unix `prog` prend un seul argument qui est un nom de fichier ou bien zéro argument et dans ce cas utilise l'entrée standard en supposant qu'elle est en accès direct.

Écrire un programme `wrap_prog` de telle façon que `wrap_prog` se comporte comme `prog`, mais fonctionne également lorsque qu'il est appelé sans argument et que l'entrée standard

n'est pas en accès direct. Pour cela, dans ce cas, il recopie l'entrée standard dans un descripteur auxiliaire en accès direct et fait ce qu'il faut pour que la commande `prog` lise dans ce descripteur.

On supposera que le programme `prog` se trouve dans le chemin d'exécution (variable d'environnement `PATH`). On supposera que le code des trois premières questions (énoncés ou réponses) a déjà été placé dans un fichier `seek.ml`.

□

Partie 4: Le coût de la concurrence

On rappelle qu'un serveur séquentiel n'accepte une nouvelle requête que lorsque la précédente est complètement traitée alors qu'un serveur concurrent accepte plusieurs requêtes en parallèle et est capable de les traiter de façon entrelacée en utilisant, par exemple, (a) plusieurs processus, (b) des coprocessus, (c) `select` et un seul processus. Dans tout cet exercice, on suppose que la machine est mono-processeur.

1) Expliquer pourquoi, bien que la machine soit mono-processeur, un serveur concurrent est souvent plus efficace.

2) Expliquer la solution (c) : on n'expliquera pas la mise en place du service mais simplement la façon dont `select` permet d'augmenter l'efficacité du serveur par rapport au serveur séquentiel.

3) Quel est, sur un ordinateur d'aujourd'hui, l'ordre de grandeur le plus proche du coût minimum d'un appel système : 0.001 μs, 1μs, 1ms, 1s ?

4) Un changement de contexte est le remplacement du processus en cours d'exécution par un autre. Donner deux raisons pour effectuer un changement de contexte.

5) Dans la suite, on notera S le coût minimal d'un changement de contexte. Ce coût est de l'ordre de 20 fois celui d'un appel système simple. Expliquer cette différence.

6) On veut comparer l'efficacité respective des modèles (b) et (c) du serveur concurrent. Pour cela, on modélise le client comme une boîte noire qui lit les réponses du serveur et lui envoie de nouvelles questions. On suppose que tous les clients se comportent de façon similaire, *i.e.* posent des questions de difficultés comparables qui demandent au serveur un temps de calcul q dans le modèle du serveur séquentiel et prennent un temps c entre une réponse et la question suivante. Typiquement ce temps est dû au temps de transmission sur le réseau entre le serveur et le client. On fera les hypothèses que $q \ll c$ et $S \ll c$.

Pour simplifier la modélisation, on supposera que les questions et réponses sont suffisamment courtes pour être lues et écrites en un seul `read` ou `write` et qu'on écrit les réponses en mode non bloquant et que l'écriture n'échoue jamais.

Enfin, on se place dans un régime stable où un nombre fixe n de clients sont (déjà) connectés. Pour n petit, le serveur attend après les clients. Pour n grand, les clients attendent après le serveur un temps supérieur au temps minimal q et on dit alors que le serveur est en *régime saturé*. On dit que le serveur est en *situation critique* lorsque la moitié des clients sont en attente après le serveur.

Avant de s'intéresser respectivement aux modèles (b) et (c), on commence par ignorer le coût de la gestion de la concurrence. ***i)*** Quelle est la valeur N de n pour laquelle le serveur

passe en régime saturé ? ***ii)*** Quelle est la valeur N' de n pour laquelle le serveur passe en situation critique ?

7) On s'intéresse maintenant au modèle à coprocessus (b) en comptant le coût de la gestion de la concurrence. ***i)*** Quelle est la valeur N_b pour laquelle le serveur passe en régume saturé ? ***ii)*** Quelle est la valeur N'_b pour laquelle le serveur passe en situation critique ? ***iii)*** Sous quelles conditions le coût de la gestion de la concurrence est-il significatif ? ***iv)*** Ces conditions vous paraissent-elles raisonnables ?

8) Le choix de la solution (a) plutôt que (b) améliorerait-il l'efficacité du serveur ?

9) On considère maintenant le modèle (c) avec `select`. L'appel système `select` est un appel système cher et son coût est nettement supérieur au coût minimal d'un appel système mais reste aussi inférieur à celui d'un changement de contexte. Pour simplifier on fera l'approximation par excès que sont coût est celui S d'un changement de contexte, S.

i) Dans le modèle avec `select`, quelle est la valeur N_c de n pour laquelle le serveur passe en régime saturé ? ***ii)*** En régime saturé, on note k le nombre de clients qui sont en moyenne en attente après le serveur. Quel est le temps nécessaire pour traiter ces k clients ? ***iii)*** En déduire la valeur N'_c de n lorsque le serveur passe en situation critique.

10) ***i)*** Quel est le modèle le plus efficace, sous les hypothèses décrites ? (on regardera les valeurs de passage en régime saturé et en particulier en régime critique.) ***ii)*** Sous quelles conditions le gain est-il significatif ?

11) Donner un exemple, pas forcément lié au modèle client serveur, où une solution similaire à (c) n'apporterait aucune amélioration par rapport à une solution séquentielle, alors qu'une solution avec coprocessus en apporterait.

□