Université Paris Diderot
Master 1 Informatique

Programmation Système
Année 2014–2015

Examen

27 mai 2015

Instructions Motivez vos réponses. Tout document papier est autorisé. Tout dispositif électronique et/ou de communication est interdit. La durée de l'examen est de 3 heures.

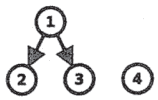
Exercice 1 [fork]

a) Un processus exécute le code C suivant :

```
fork();
if (i % 2 == 0)   fork();
if (i % 3 == 0)   fork();
fork();
/* ... */
```

Combien des *nouveaux* processus crée l'exécution du code ci-dessus quand `i` vaut, respectivement : 2, 3, et 6 ?

b) Écrivez le code C nécessaires pour créer un ensemble de processus correspondant à la hiérarchie *processus pères* $\longrightarrow$ *processus fils* suivante :



Notez que le processus 4 *n'est pas* en relation de parentèle avec les autres processus dans la hiérarchie, mais doit être crée par votre code.

Exercice 2 [pipe]

Écrivez un programme C qui émule le comportement d'un shell lors de l'exécution de :

```
grep howdy < foo.txt | wc -l > results.txt
```

votre programme doit créer un processus pour chaque sous-commande, lier ensemble les processus avec de tubes anonymes, implémenter les rédirections d'entrée/sortie, exécuter les programmes externes avec les arguments appropriés, et terminer avec la valeur de retour de `wc`. Si l'utilisateur frappe `CTRL-C` ou `CTRL-\`, le signal correspondant doit être livré à toutes les sous-commandes en exécution.

Exercice 3 [mmap]

a) Écrivez un programme C `multimv` avec le prototype suivant (les arguments de `DEST2` à `DESTn` sont optionnels) :

```
multimv SOURCE DEST1 [ DEST2 ... DESTn ]
```

`multimv` doit avoir le même effet (mais idéalement une meilleur efficacité) de la commande shell : `mv SOURCE DEST1 ; cp DEST1 DEST2 ; ... ; cp DEST1 DESTn`. `multimv` doit utiliser la technique de projection en mémoire (`mmap`) pour la lecture du fichier `SOURCE`.

b) Pensez-vous que `multimv` soit un cas d'usage approprié pour la technique de I/O basée sur projection en mémoire ? Pourquoi ?

Exercice 4 [signaux + multiprocessing]

a) Écrivez un programme C `count0 FILE` qui fork 2 processus fils : le premier va travailler sur la première moitié du fichier `FILE`, l'autre sur la deuxième moitié. Chaque processus fils lit sa propre partie de `FILE` et envoie un signal au processus père pour chaque octet qui vaut 0 rencontrée. Une fois terminé le traitement du fichier, le processus père affiche sur l'écran le nombre des octets 0 (totales) rencontrés, et termine.

b) Votre implémentation garantie que le résultat affiché sur l'écran correspond aux octets 0 contenus dans `FILE` ? Si oui, expliquez pourquoi ; si non, expliquez comment corriger le problème.

Exercice 5 [socket UNIX]

Écrivez en C un simple serveur local de *streaming* des fichiers MP3 et son client.

Le serveur a comme prototype `streamp3 MUSICDIR`, où `MUSICDIR` est le directory racine qui contient les fichiers musicales. Le client a prototype `getmp3 PATH..`, où `PATH` est un parcours, relative à `MUSICDIR`, qui point vers un ou plusieurs fichiers avec extensions `.mp3`.

Le protocole de communication entre client et serveur prévoit l'usage d'un seul socket UNIX de type stream. Le client se connecte au socket et envoie un `PATH`, terminé par `\n`. Le serveur répond en envoyant d'abord la taille du fichier (un entier de type `off_t`, comme retourné par `stat`) et ensuite le contenue du fichier. En cas d'erreur (p.ex., si le fichier n'existe pas), le serveur envoie -1 aux client et termine la connexion. En cas des plusieurs `PATH`, le client envoie le prochain path au serveur, sans interrompre la connexion, une fois terminé la réception du fichier précèdent.

Dans cette version préliminaire du projet, le client envoie sur la sortie standard le contenu du fichier MP3 réçu (mais pas sa taille !). Cela permettra de le brancher, p.ex., avec une pipe, à un véritable player MP3.

Idéalement, le serveur doit pouvoir supporter plusieurs clients en parallèle, sans attentes entre eux.