# M2 Systèmes Avancés

## *Examen du 25 Mars 2011 8H30 - 11H30*

## *Documents autorisés*

Le total des points est de 60. Votre note finale sera obtenue en divisant vos points par 3.

## *1 Interfaces (6 points)*

1. *(2 points)* Décrivez brièvement ce que définit le standard LSB et quelle est son utilité. Donnez la signification de cet acronyme.
2. *(4 points)* Qu'est-ce qui permet de porter une application d'un système à un autre sans re-compilation ni édition de liens? Cela permet-il d'exécuter une application (binaire) générée pour une machine Linux/x86 sur un Linux/Sparc? Pourquoi? Qu'est ce qui permet de porter une application d'un système à un autre après re-compilation et édition de liens? Cela permet-il d'exécuter une application (binaire) générée pour une machine Linux/x86 sur un Linux/Sparc? Pourquoi?

## *2 Gestion de processus (24 points)*

3. *(4 points)* Quelle est l'utilité des « listes de hash »? Décrivez leur principe de fonctionnement.
4. *(6 points)* Sur un système Linux, est-il possible de créer un processus différent de « `init` » qui recevra la valeur « `1` » en retour d'un appel système `getpid()`? Si oui, comment peut-on créer un tel processus? Quel valeur recevra-t-il en retour d'un appel à `getppid()`? Peut-on créer plusieurs processus ayant ce même pid « `1` »? Peuvent-ils communiquer entre eux par l'intermédiaire de tubes nommés? Si non, pourquoi? Si oui, y a-t-il un moyen de les empêcher de communiquer par tubes nommés?
5. *(3 points)* De la même manière, est-il possible sur une machine Linux de créer simultanément deux shells obtenant des valeurs différentes en retour à l'exécution de la commande « `uname -a` »? Pourquoi? Comment?
6. *(2 points)* Quelle est l'utilité des mécanismes ci-dessus? Par quel(s) autres mécanisme(s) peuvent-ils / doivent-ils être compléter pour être réellement utiles?
7. *(4 points)* Comment le noyau Linux fait-il pour retrouver le contexte système d'un processus lorsque ce processus effectue un appel système, par exemple fork()?
8. *(5 points)* Sur un système Linux, on suppose que l'on a seulement deux processus (A et B) qui se synchronisent grâce à un tube nommé, initialement vide. La séquence d'exécution est la suivante:

   8.1. Le processus A s'exécute. Le processus B est dans un appel système « pause() ».

   8.2. Le processus A envoie un signal au processus B.

   8.3. Le processus A effectue la lecture d'une donnée sur le tube nommé.

   8.4. Le processus B écrit une donnée dans le tube nommé, puis refait un appel système « pause() »

   8.5. Le processus A récupère la donnée.

   8.1. On recommence en 8.1.

   Décrivez pour les étapes 8.1. à 8.5. l'état dans le quel se trouvent chacun des 2 processus vis-à-vis de l'ordonnanceur. Pour chaque étape, on donnera deux états. On expliquera les raisons des modifications d'état.

## 3 Gestion mémoire (10 points)

9. *(4 points)* L'allocateur mémoire de base du noyau Linux est basé sur sur mécanisme dit de « buddy-system ». Ce « buddy-system » a 11 niveaux. *(4 points)*

   9.1. Quelle taille mémoire maximum peut-être allouée en un seul appel à cet allocateur?

   9.2. Quelle est la plus petite taille mémoire que l'on peut lui demander?

   9.3. Soit une demande d'allocation d'un zone mémoire de 4 pages. Parmi les adresses suivantes, quelle pourrait être l'adresse de la zone allouée? Pourquoi?

   a) 0xD010 C000 b) 0xD010 5000 c) 0xD010 0800

10. *(6 points)* Soit l'exécution des commandes suivantes ainsi que leurs résultats.

```
# ls -li execl toto
278324 -rwxr-xr-x 2 francois francois 570341 2011-03-12 12:20 execl
278324 -rwxr-xr-x 2 francois francois 570341 2011-03-12 12:20 toto

# file execl toto
execl: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
statically linked, for GNU/Linux 2.6.8, not stripped
toto:  ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
statically linked, for GNU/Linux 2.6.8, not stripped

# size execl toto
   text      data       bss       dec       hex   filename
 494730      1928      8408    505066      7b4ea   execl
 494730      1928      8408    505066      7b4ea   toto
```

Quelle taille maximum théorique l'exécution simultanée de ces 2 programmes occuperait-elle en mémoire? Explicitez vos calculs. On ne tiendra compte que des sections de code et de données. On ignorera les piles et autres régions créées dynamiquement par le système.

On suppose que le processus exécutant « `execl` » fait un `fork` et que simultanément, le processus exécutant « `toto` » fait un `vfork`. Quelle est la taille mémoire maximum théorique utilisée par ces 4 processus? Pourquoi?

## 4 Gestion de Fichiers (20 points)

11. *(2 points)* Quelle commande Linux/Unix a-t-il fallu effectuer au préalable pour que la commande ls de la question 10. ci-dessus affiche ce résultat?

12. *(4 points)* Quels sont les mérites respectifs des configurations de disques RAID1, RAID3 et RAID5?

13. *(6 points)* Quelle est la structure générique d'un système de fichiers sur disque? Il n'est pas demandé la structure détaillée d'un système de fichiers particuliers, mais la structure générale: zones et objets stockés sur un disque.

14. *(3 points)* Quel problème lié à NFS est-il résolu par le « numéro de génération » que l'on trouve dans certains systèmes de fichiers?

15. *(5 points)* Qu'est-ce qu'une opération idempotente? Et une opération non-idempotente? Les appels systèmes `write(int fd, void* buf, size_t lg)` et `pwrite (int fd, void *buf, size_t nbyte, off_t offset)` sont-ils idempotents? Donnez un exemple d'opération NFS qui ne soit pas idempotente.