Examen du 18 octobre 2013 (1h)

RS : Réseaux et Systèmes

La notation tiendra compte de la validité des réponses, mais aussi de la présentation et de la clarté de la rédaction. Lisez entièrement le sujet avant de commencer calmement.

Documents interdits, à l'exception d'une feuille A4 recto-verso manuscrite.

★ **Exercice 1: Question de cours (6pts)** (d'après Raymond Namyst).

▷ **Question 1:** Qu'est ce que l'attente active ?

▷ **Question 2:** Qu'est ce qu'un i-node (ou i-nœud) ?

▷ **Question 3:** Rappelez brièvement le principe de fonctionnement d'un appel système (sur un exemple concret de votre choix). Un petit schéma explicatif est bienvenu.

▷ **Question 4:** Quel est l'intérêt du mécanisme des appels systèmes dans un système d'exploitation multi-utilisateurs ? Pourquoi ne pas utiliser de simples appels de fonctions normaux à la place ?

▷ **Question 5:** Pourquoi est-on sûrs qu'aucune application ne peut contourner le mécanisme des appels systèmes par aucun moyen que ce soit une fois que le système d'exploitation a démarré ?

★ **Exercice 2: Relire des synchronisations (7pts)** (d'après Françoise Baude).

Voici ci-dessous une variante d'une synchronisation de type Producteurs / Consommateur.

▷ **Question 1:** Décrire **clairement** quelles sont les synchronisations mises en œuvres lorsqu'on exécute ce pseudo-code avec N producteurs et un unique consommateur. Donnez l'idée d'un scénario possible d'exécution, en indiquant bien quand certaines actions doivent se passer avant d'autres.

Une explication claire, complète et concise est naturellement préférable à un texte long et confus.

INDICATION : Ce code ne présente aucune condition de compétition ni d'interblocage (sauf erreur).

```
1  Déclaration et initialisation de variables globales
2    tour: variable entière initialisée à 0
3    tampon: tableau d'objets de capacité N où sont stockés les objets produits
4    consomme: sémaphore initialisée à 0
5    mutex: sémaphore initialisée à 1
6
7  Procédure du processus Producteur numéro i {
8    variable locale : objet
9
10   objet = fonction_de_production() // produire un objet, sa nature importe peu ici
11   P(mutex)
12   tampon[tour] <- objet
13   tour <- tour + 1
14   V(mutex)
15   V(consomme)
16 }
17 Procédure du processus Consommateur { // ce processus est unique
18   Répéter N fois {
19     P(consomme)
20   }
21   Pour i prenant les valeurs entre 1 et N {
22     affichage(tampon[i])
23   }
24 }
```

▷ **Question 2:** Discutez les avantages et inconvénients par rapport au modèle classique de producteurs/consommateur.

▷ **Question 3:** Que se passe-t-il si on inverse les lignes 10 et 11 du pseudo-code fourni ?

```
10   P(mutex)
11   objet = fonction_de_production() // produire un objet, sa nature importe peu ici
```

▷ **Question 4:** Que se passe-t-il si on inverse (seulement) les lignes 13 et 14 du pseudo-code fourni ?

```
13   V(mutex)
14   tour <- tour + 1
```

▷ **Question 5:** Que se passe-t-il si on inverse (seulement) les lignes 14 et 15 du pseudo-code fourni ?

```
14   V(consomme)
15   V(mutex)
```

★ Exercice 3: Relire des tubes (7pts).

▷ **Question 1:** Dessinez les relations entre les différents processus (sous forme de patates) et tubes (sous forme de liens entre les patates) créés par le programme ci-contre.

▷ **Question 2:** Sachant que la macro `islower()` retourne vrai si le caractère passé en argument est en minuscule et faux si ce caractère est en majuscule, donnez les grandes lignes de ce qui se passe lorsqu'on exécute ce programme de la façon suivante :

```
$ gcc -Wall -o mystere mystere.c
$ echo BonJOUr | ./mystere SaLuT
```

▷ **Question 3:** Raffinez votre réponse en discutant les différents ordres d'affichage possibles avec la commande suivante :

```
$ gcc -Wall -o mystere mystere.c
$ echo abAB | ./mystere abAB
```

▷ **Question 4:** Que se passe-t-il si l'on supprime l'ensemble des lignes 24 à 27 ? Pourquoi ?

```
24 // close(A[1]);
25 // close(B[1]);
26 // wait(NULL);
27 // wait(NULL);
```

▷ **Question 5:** Que se passe-t-il si l'on commente les lignes 24 et 25 de ce programme en restaurant les lignes 26 et 27 ? Pourquoi ?

```
24 // close(A[1]);
25 // close(B[1]);
26 wait(NULL);
27 wait(NULL);
```

mystere.c

```
1  #include <stdio.h>
2  #include <ctype.h> // islower()
3  #include <unistd.h>
4  #include <sys/wait.h>
5
6  int main(int argc, char *argv[]){
7    int A[2], B[2];
8    char c;
9
10   pipe(A);
11   pipe(B);
12
13   if (fork()) {
14     if (fork()) {
15       close(A[0]);
16       close(B[0]);
17       while (read(0,&c,1) == 1) {
18         if (islower(c)) {
19           write(A[1], &c, 1);
20         } else {
21           write(B[1], &c, 1);
22         }
23       }
24       close(A[1]);
25       close(B[1]);
26       wait(NULL);
27       wait(NULL);
28     } else {
29       dup2(B[0],0);
30       close(A[0]);
31       close(A[1]);
32       close(B[0]);
33       close(B[1]);
34       while (read(0,&c,1) == 1)
35         printf("1: %c\n", c);
36     }
37   } else {
38       dup2(A[0],0);
39       close(A[0]);
40       close(A[1]);
41       close(B[0]);
42       close(B[1]);
43       while (read(0,&c,1) == 1)
44         printf("2: %c\n", c);
45   }
46   return 0;
47 }
```