Examen du 10 octobre 2018 (1h)

RS : Réseaux et Systèmes

Deuxième année

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 à rendre avec votre copie.

★ Exercice 1: Questions de cours (2 pts)

▷ **Question 1** (1 point)**:** Expliquez précisément ce que fait l'appel système `fork`.

▷ **Question 2** (1 point)**:** Expliquez précisément ce que fait l'appel système `exec`.

★ Exercice 2: Programmation concurrente, d'après Luigi Santocanale (5pts)

▷ **Question 1** (2 points)**:** Rappelons que l'appel système `int mkdir(const char *path , mode_t mode);` retourne un code d'erreur si le répertoire à créer existe déjà. Expliquez, par conséquence, comment par l'implantation suivante des fonctions `entrer_region` et `sortir_region`, on peut protéger la section critique d'un processus et assurer l'exclusion mutuelle :

```
1 void entrer_region(char *verrou) {
2   while(mkdir(verrou ,0777) < 0) {
3     sleep(1);
4   }
5 }
```

```
1 void sortir_region(char *verrou) {
2   rmdir(verrou);
3 }
```

▷ **Question 2** (1 point)**:** À quel type d'attente cette implémentation s'apparente-t-elle ?

▷ **Question 3** (2 points)**:** Dans le code suivant `verrou` est l'adresse d'une variable entière que l'on suppose être partagée entre plusieurs processus. On peut alors protéger la section critique d'un processus par

```
1 void entrer_region(int *verrou) {
2   while (*verrou == 1) {
3     sleep(1);
4   }
5   *verrou = 1;
6 }
```

```
1 void sortir_region(int *verrou) {
2   *verrou = 0;
3 }
```

Expliquez en quoi cette implantation est pire que la précédente.

★ Exercice 3: Threads (2 pts) – Étudiants de l'École des Mines uniquement

Lors du TP3, un étudiant a écrit le Programme T ci-dessous. L'objectif de l'étudiant est le suivant : créer 10 threads (numérotés de 1 à 10), puis chaque thread ajoute 1000000 fois son numéro à une variable somme. Enfin le programme affiche la valeur de la somme (55000000 si tout va bien).

▷ **Question 1** (0.5 point)**:** Que pouvez-vous dire des différents affichages de `getpid()` ? Pourquoi ?

▷ **Question 2** (1 point)**:** L'étudiant a fait deux erreurs qui provoquent un résultat complètement faux. Quelles sont-elles ? (Il n'est pas demandé d'écrire un programme correct, mais uniquement d'expliquer précisément quelles sont les deux erreurs et comment on pourrait les corriger)

★ Exercice 4: Signaux (2pts) – Étudiants de TELECOM Nancy uniquement

Expliquez précisément le programme S ci-dessous.

Programme S

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void f2(int sig) {
    printf("bar\n");
    exit(0);
}

int compt = -1;

void f1(int sig) {

    if (++compt == 3) {
        kill(getpid(), SIGUSR1);
    } else {
        printf("foo\n");
    }
}

int main() {
    struct sigaction nvt, old;
    memset(&nvt,0,sizeof(nvt));
    nvt.sa_handler = f1;
    sigaction(SIGINT,  &nvt, &old);
    nvt.sa_handler = f2;
    sigaction(SIGUSR1,  &nvt, &old);
    while(1);
}
```

Programme T

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/types.h>
#include <unistd.h>

#define NBTH 10

int somme = 0;

void * ajouteur(void * arg) {
  int numthread = * (int*) arg;
  int i;
  int s;
  printf("ajouteur: getpid()=%d\n",
         getpid());
  for (i = 0; i < 1000000; i++) {
    s = somme;
    s += numthread;
    somme = s;
  }
  return NULL;
}

int main(int argc, char** argv) {
  int i;
  pthread_t threads[NBTH+1];
  printf("main: getpid()=%d\n",
         getpid());
  for (i = 1; i <= NBTH; i++) {
    pthread_create(
       &threads[i], NULL,
             ajouteur, (void*) &i);
  }
  for (i = 1; i <= NBTH; i++) {
    pthread_join(threads[i], NULL);
  }
  printf("somme=%d\n", somme);
  return 0;
}
```

★ Exercice 5: Processus et tubes (6pts).

▷ **Question 1:** Dans le programme ci-contre, indiquez quels sont les processus créés (et à quelle ligne), ainsi que les lignes exécutées par chaque processus.

▷ **Question 2:** Dessinez les relations entre les différents processus et tubes créés.

▷ **Question 3:** 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 4:** 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 5:** 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 6:** 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 }
```

★ **Exercice 6: Savoir reconnaître les schémas de synchronisation classiques et utiliser les sémaphores (5 pts)** (inspiré d'un exercice de Cédric Bastoul)

Un étudiant qui se spécialise en anthropologie et accessoirement en informatique s'est embarqué dans un projet de recherche pour voir s'il était possible d'enseigner les interblocages aux babouins d'Afrique. Il repère un profond canyon et y jette une corde au travers, de sorte qu'un babouin puisse le traverser à bouts de bras.

Chaque babouin répète l'algorithme suivant :

- RÉPÉTER à l'infini
  - Se présenter devant la corde
  - Attraper la corde
  - Traverser le canyon
  - Relâcher la corde
  - Continuer à pieds
- FIN RÉPÉTER

L'installation ayant du succès, les babouins se rendent vite compte que si deux d'entre eux commencent à traverser dans les deux sens opposés, ils se retrouvent bloqués au milieu du canyon.

▷ **Question 1:** Proposez une solution (modifiez l'algorithme ci-dessus) à base de sémaphore garantissant que seul un babouin à la fois pourra accéder à la corde et traverser.

▷ **Question 2:** Votre solution se rapproche d'un schéma de synchronisation classique. Lequel ?

Cette solution est loin d'être idéale. Les babouins se rendent rapidement compte que plusieurs babouins pourraient traverser simultanément, à condition qu'ils traversent tous dans le même sens, ce qui serait bien plus efficace.

▷ **Question 3:** Modifiez l'algorithme ci-dessus. Il est conseillé d'écrire deux algorithmes : celui pour les babouins qui traversent du Nord vers le Sud (BabouinNS), et celui pour ceux qui font l'inverse (BabouinSN).

▷ **Question 4:** Votre solution se rapproche d'un schéma de synchronisation classique. Lequel ?

▷ **Question 5:** Que pouvez-vous dire d'un problème de famine éventuel avec votre solution ? (Il n'est pas demandé de proposer obligatoirement une solution sans famine)