# Examen du 14 octobre 2010 (1h30)

RS : Réseaux et Systèmes
1A, 2A ou 3A

La notation tiendra compte de la validité des réponses, mais aussi de la présentation et de la clarté de la rédaction.

**Documents interdits, à l'exception d'une feuille A4 à rendre avec votre copie.**

★ **Exercice 1: Écrire des fork, des exec et des wait** (3pts)

Écrivez un programme permettant d'exécuter en parallèle un ensemble de programmes donné en paramètres (eux même sans paramètres) et qui s'arrête après la terminaison de tous. Le code de retour de la commande signalera un échec si et seulement si l'une des commandes termine en échec. Le prototype de la commande est : `parallele c1 c2 ... cn`

★ **Exercice 2: Lire des fork, des exec et des wait** (3pts)

Considérez le programme ci-contre.

▷ **Question 1:** Dessinez le schéma de filiation entre processus obtenu.

▷ **Question 2:** Quel est l'effet de la ligne 9 ?

▷ **Question 3:** Que se passe-t-il si l'on remplace la ligne 7 par `execvp(argv[0],argv)` ?

Programme 1

```
1  static int n =3;
2  main (int argc, char*argv[]) {
3    int i ;
4    for (i =0; i < n ; i ++)
5      if (fork ()==0) {
6        n --;
7        main(argc,argv);
8      }
9    while ( wait ( NULL )!= -1);
10   exit ( EXIT_SUCCESS );
11 }
```

★ **Exercice 3: Lectures/écritures sur disque** (2pts – d'après Luigi Santocanale)

Considérez les deux programmes suivants :

cp1.c

```
1  int main ( int argc , char * argv []) {
2     int d1 , d2 ;
3     char c ;
4
5     d1 = open ( argv [1] , O_RDONLY );
6     d2 = open ( argv [2] , O_WRONLY | O_CREAT | O_TRUNC,
7                 S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH );
8     while ( read ( d1 ,& c ,1)==1 )
9        write ( d2 ,& c ,1);
10    close ( d1 );
11    close ( d2 );
12    exit ( EXIT_SUCCESS );
13 }
```

cp2.c

```
1  int main ( int argc , char * argv []) {
2     FILE * d1 , * d2 ;
3     char c ;
4
5     d1 = fopen ( argv [1] ,"r");
6     d2 = fopen ( argv [2] ,"w+");
7     while (!feof(d1)) {
8        fscanf(d1,"%c",&c);
9        fprintf (d2,"%c", c);
10    }
11    fclose (d1);
12    fclose (d2);
13    exit ( EXIT_SUCCESS );
14 }
15
```

Voici les temps d'exécution de chacun d'eux alors que le fichier f est de taille 1752064 octets :

| commande | temps utilisateur | temps système |
|---|---|---|
| cp1 f g | 9s | 53s |
| cp2 f g | 1s | 0,06s |

▷ **Question 1:** Justifiez ces mesures.

*(Tournez la page svp)*

★ **Exercice 4: Des signaux.** (2pts – d'après Jean-Baptiste Yunès)

Considérez le programme ci-contre.
Son exécution produit le résultat suivant :

```
351 a envoye 1000000 signaux
351 a recu 1000000 signaux
352 a recu 40699 signaux
```

▷ **Question 1:** Expliquer pourquoi le processus 352 n'a visiblement reçu que 40699 signaux.

▷ **Question 2:** Décrivez comment cette application termine.

```
#define N 1000000
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static int compteur, jesuislefils;

void affichage_final(int sig) {
   printf("%d a recu %d signaux\n",getpid(),compteur);
   if ( ! jesuislefils)
     wait(NULL);
   exit(EXIT_SUCCESS);
}

void piege(int sig) {
   int i;
   compteur++;
   if (jesuislefils)
     for (i=0; i<10000; i++);
}

int main(int argc,char *argv[]) {
   pid_t pid, moi;
   int i;
   struct sigaction sigact;

   memset(&sigact, 0,sizeof(sigact));
   sigact.sa_handler = piege;
   sigaction(SIGUSR1,&sigact,NULL);

   sigact.sa_handler = affichage_final;
   sigaction(SIGUSR2,&sigact,NULL);

   pid=fork();
   if (pid==0) {
      jesuislefils = 1;
      while (1) pause();
      exit(EXIT_SUCCESS);
   } else {
      jesuislefils = 0;
      moi = getpid();
      for (i=0; i<N; i++) {
         kill(pid,SIGUSR1);
         kill(moi,SIGUSR1);
      }

      printf("%d a envoye %d signaux\n",getpid(),N);
      kill(pid,SIGUSR2);
      kill(moi,SIGUSR2);
   }
}
```

★ **Exercice 5: Le parc jurassique.** (5pts – d'après un exercice de l'université Paris Diderot en 2005)

L'objet de cet exercice est de réaliser une simulation du fonctionnement d'un parc d'attractions.

Le parc jurassique est composé d'un musée des dinosaures et d'un parc pour safari. On considère qu'il y peut arriver des visiteurs de manière aléatoire en nombre et en temps. Tout visiteur commence par visiter le musée et cette visite dure un temps variable. Après cette visite, un visiteur fait une randonnée-safari dans le parc. Pour cela, il doit monter dans une voiture. Il y a un seul visiteur par voiture et le nombre de voitures est fixe : cela nécessite donc une attente. Lorsqu'une voiture est disponible, un passager unique y embarque et lorsqu'il est installé, la voiture s'en va. Le parcours suivi et la durée de la promenade sont variables. Le passager est passif. Ce n'est pas lui qui fixe la durée du safari mais la voiture.

```
Visiteur
visite du musee des dinosaures
attente de la voiture
embarquement dans la voiture
safari (c'est la voiture qui
     indique la fin du safari)
descente de la voiture
```

```
Voiture
Faire a l'infini :
   attendre qu'un visiteur ait embarque
   faire le safari
   indiquer au visiteur la fin du trajet
   attendre que le visiteur ait debarque
```

On désigne par `nb_voitures` le nombre de voitures, par la fonction `visite()` la visite du musée et par la fonction `safari()` la promenade dans le parc.

▷ **Question 1:** Écrivez (en utilisant des variables partagées et des sémaphores) un algorithme pour un visiteur quelconque et un algorithme pour la voiture en considérant qu'il n'y a qu'une seule voiture dans le parc.

▷ **Question 2:** De quel problème classique cet exercice est-il inspiré ?

▷ **Question 3:** Adaptez votre solution au cas où le parc dispose de plusieurs voitures. On souhaite que les voitures roulent toujours dans le même ordre sur la piste, et donc que chaque passager à son tour monte dans la première voiture de la file d'attente.

▷ **Question 4:** Adapter votre solution au cas où chaque voiture a une capacité propre (la voiture $i$ peut

accueillir $c_i$ passagers) et ne part que quand elle est pleine.