Examen du 15 octobre 2012 (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.

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

★ Exercice 1: Lire des `fork` (5pts)

Faire un schéma représentant les différents processus créés par le Programme 1 ci-dessous, avec leurs appels à printf() et à exit().

★ Exercice 2: Savoir utiliser les threads POSIX (3pts)

Lors du TP3, un étudiant a écrit le Programme 2 ci-dessous.

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

▷ **Question 2:** L'étudiant a fait deux erreurs grossières. Quelles sont-elles ? (Il n'est pas demandé d'écrire un programme correct, mais uniquement d'expliquer précisément quelles sont les deux erreurs)

Programme 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    int i, status;
    for (i = 0; i < 2; i++)
    {
        if (fork() != 0)
        {
            printf("a");
            if (fork() == 0) {
                printf("c");
            } else {
                printf("d");
                exit(0);
            }
        } else {
            printf("b");
        }
    }
    exit(42);
}
```

Programme 2

```c
#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;
    printf("ajouteur: getpid()=%d\n", getpid());
    for (i = 0; i < 1000000; i++) {
        somme += numthread;
    }
    return NULL;
}

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

TSVP

★ Exercice 3: Utiliser `fork`, `exec`, `wait`, `waitpid`, `pipe`, `dup`, `dup2` (6pts)

Donner le code C (sans utiliser la fonction `system`) correspondant à ce que fait un *shell* lorsqu'on tape les lignes de commandes suivantes. Afin d'obtenir un code lisible et court, vous êtes dispensés des tests d'erreur des appels systèmes.

▷ **Question 1:** `prog1 && prog2`

(Il faut donc : lancer un programme *prog1*, attendre sa fin, récupérer son code de retour, et si ce code de retour est égal à 0, lancer un programme *prog2* et attendre sa fin)

▷ **Question 2:** `prog1 | prog2`

(Il faut donc : lancer deux programmes *prog1* et *prog2*, en liant la sortie standard de *prog1* à l'entrée de *prog2*)

Extraits de quelques pages de manuel

```
pid_t fork(void);
int execlp(const char *file, const char *arg, ...);
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
WIFEXITED(status) -- returns true if the child terminated normally
WEXITSTATUS(status) -- returns the exit status of the child
int pipe(int pipefd[2]);
int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

★ Exercice 4: Savoir reconnaître les schémas de synchronisation classiques et utiliser les sémaphores (6pts)

L'infâme pirate Barbe-Noire et ses 3 non moins infâmes lieutenants (LeBorgne, Long Jean d'Argent et Rackham le Rose) fêtent avec force rhum la prise d'un magnifique galion espagnol. Chacun des 4 hommes se met à avoir le comportement suivant :

- RÉPÉTER à l'infini
  - Essayer d'entrer dans la cale pour admirer le trésor
  - Admirer le trésor dans la cale pendant T_a secondes
  - Sortir de la cale
  - Cuver son rhum pendant T_c secondes
- FIN RÉPÉTER

(les valeurs de T_a et T_c importent peu)

▷ **Question 1:** Les pirates étant plutôt solitaires, ils préfèrent être seuls pour admirer le trésor. Proposez une solution (écrivez l'algorithme `Pirate`) à base de sémaphore garantissant que seul un pirate pourra entrer dans la cale.

▷ **Question 2:** De quel schéma de synchronisation ce problème se rapproche-t-il ?

La soirée avançant, les pirates relâchent les règles, et décident que :

- Barbe-Noire doit être seul quand il contemple le trésor :
  - Lorsque Barbe-Noire admire le trésor, si un de ses lieutenants essaye d'entrer dans la cale, il attend que Barbe-Noire en soit sorti avant d'entrer à son tour.
  - Si Barbe-Noire veut entrer dans la cale alors qu'un (ou plusieurs) de ses lieutenant(s) admire(nt) le trésor, Barbe-Noire attend qu'il n'y ait plus personne dans la cale avant d'entrer à son tour.
- Les lieutenants peuvent admirer le trésor ensemble :
  Ainsi si un lieutenant est en train d'admirer le trésor dans la cale et qu'un autre lieutenant essaye d'entrer dans la cale, cet autre lieutenant entre dans la cale sans attendre.
- Personne ne double Barbe-Noire :
  Si un ou plusieurs lieutenants admirent le trésor et que Barbe-Noire attend d'entrer dans la cale, les autres lieutenants qui auraient envie d'admirer le trésor doivent attendre que Barbe-Noire soit entré dans la cale, ait admiré le trésor et soit sorti de la cale avant d'entrer à leur tour.

▷ **Question 3:** De quel schéma de synchronisation ce problème se rapproche-t-il maintenant ?

▷ **Question 4:** La règle « *Personne ne double Barbe-Noire* » permet d'éviter un problème courant. Quel est le nom de ce problème ? Que pourrait-il se produire si cette règle n'était pas présente ?

▷ **Question 5:** Implémentez les processus `BarbeNoire` et `Lieutenant` respectant les règles ci-dessus grâce à des sémaphores.