info507 : Systèmes d'exploitation
TD 1 : appels système, processus

Pierre Hyvernat, François Boussion, Gérald Cavallini
Pierre.Hyvernat@univ-smb.fr
Francois.Boussion@univ-smb.fr
Gerald.Cavallini@univ-smb.fr

Exercice 1 : Appels système

Question 1.
- Rappelez la définition d'appel système.
- Comment savoir si une fonction C provoque des appels système ?
- Est-il important de savoir quelles fonctions provoquent des appels système ?

Question 2. Parmi les fonctions suivantes, quelles sont celles qui provoquent des appels système ?
- `strcmp()`, pour comparer des chaines de caractères,
- `malloc()`, pour allouer de la mémoire dans un processus,
- `fork()`, pour dupliquer un processus,
- `execv()`, pour remplacer le processus courant par un autre,
- `memcpy()`, pour copier une zone de mémoire vers une autre,
- `kill()`, pour envoyer un signal à un autre processus,
- `random()`, pour obtenir un nombre aléatoire,
- `printf()`, pour faire de l'affichage.

Exercice 2 : Les états possibles d'un processus

Question 1. Lors de sa vie, un processus peut se trouver dans plusieurs états :
- en exécution,
- prêt,
- bloqué,
- terminé.

Donnez des exemples des situations de processus dans chacun de ces états et détaillez les transitions possibles.

Question 2. L'ordonnanceur Linux ne considère pas "en exécution" comme un état différent de "prêt", et possède plusieurs états supplémentaires. Par exemple :
- `TASK_STOPPED` pour les processus en pause,
- `TASK_ZOMBIE` pour les processus zombie.

Dans quels cas des processus peuvent se retrouver dans ces états ? Quelles transitions faut-il ajouter pour les incorporer au graphe précédent ?

Question 3. Pour stopper un processus quelques secondes, il est préférable d'utiliser une attente "passive" (appel système `sleep`) plutôt qu'une attente "active" (boucle `while` suffisamment longue).

Pourquoi ?

Dans quel cas est-ce qu'une attente passive est impossible ?

Exercice 3 : Création de processus

Question 1. Rappelez le fonctionnement de l'appel système `fork`.

Écrivez une fonction C qui créé un processus et provoque un affichage du type

```
Je suis le père, mon fils a le PID 383.
Je suis le fils.
```

Question 2. Écrivez deux fonctions avec un paramètre `n` qui génèrent des processus avec les arborescences suivantes

```
processus
 |- fils-1
 |- fils-2
 |- fils-3
 ...
 |- fils-n
```

```
processus
 |- fils-1
     |- fils-2
         |- fils-3
              ...
                |- fils-n
```

Question 3. Quelle sera l'arborescence générée par le programme suivant

```
for (int i=0; i<4; i++) {
    fork();
}
while();
```

Question 4. L'appel système `execv` permet de *remplacer* le processus courant par au autre. Son prototype est

```
int execv(const char *path, char *const argv[]);
```

(et ses variantes) a une seule valeur de retour possible. Que représente t'elle ?

Question 5. Un shell permet de lancer des processus depuis un terminal. Son code contient donc quelque chose comme

```
int p = fork();
if (p == 0) {
    execv(commande, ...);
}
```

Expliquez à quel endroit doivent être gérées le *redirections* de `stdin`, `stdout` et `stderr`.

Exercice 4 : une autre utilisation de `fork`

Question 1. Quel est le comportement attendu du programme suivant ?

```
#include <stdio.h>
#define N 12345
int main(int argc, char** argv)
{
    int T[N];
    printf("DÉBUT TEST\n");
    int i = 0;
    while (1) {
        printf("%d -> %d\n", i, T[i]);
        if (T[i] == -1234) break;
        i++;
    }
    printf("FIN TEST\n");
    // ... autres trucs ...
    return 0;
}
```

Question 2. Comment peut-on garantir que ce programme ne provoque pas d'erreur de segmentation ?