TP N°4
Création de processus

Sylvain Chevallier

Préambule

Notions : Utilisation des fonctions `fork()` et `exec()`

Prochain TD : Synchronisation entre processus

Rappel

```c
#include <unistd.h>
pid_t fork(void);
```

La primitive `fork()`, appellé dans un processus père, crée un processus fils. Les deux processus partagent le même code. Pour pouvoir les différencier, `fork()` renvoie `0` dans le processus fils et renvoie le PID du fils dans le processus père.

```c
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

La primitive `exec` permet une approche alternative par rapport à `fork()` : l'exécutable appelé par `exec` (`path` ou `file` dans les prototypes) remplace les instructions du processus père. Les arguments sont passés en liste avec `execl` et en bloc avec `execv`. Lors de l'appel à `execlp` ou `execvp`, l'exécutable à lancer est recherché dans le `PATH`.

```c
pid_t getpid(void);
pid_t getppid(void);
```

Ces fonctions renvoient respectivement le PID du processus qui les appelle ou le PID de leur processus père (PPID).

```c
unsigned int sleep(unsigned int seconds);
```

La fonction `sleep()` permet de suspendre un processus pendant un nombre de seconde passée en argument.

1 Création de processus

Donner en les justifiant les affichages obtenus sur la sortie standard pour le programme ci-dessous.

```
1           #include <stdlib.h>
            #include <stdio.h>
3           #include <sys/types.h>
            #include <unistd.h>
5   int main() {
       pid_t idproc;
7 1     idproc= fork();
  2
9 3     if (idproc ==-1) {
  4        perror ("echec du fork");
11 5        exit(EXIT_FAILURE);
        }
13 6     //printf ("A- Processus %d\n",getpid()); // à ajouter pour question b)
  7     if (idproc ==0) {
15 8        printf("processus fils %d\n",getpid());
  9        exit (EXIT_SUCCESS);  // à enlever pour question b)
17      }
  10      printf("B-Processus %d\n", getpid());
19 11      exit(EXIT_SUCCESS);
    }
```

b) Même question en ajoutant la ligne 6 et en enlevant la ligne 9

2 Partages des variables

Donner, en les justifiant, les affichages produits par le lancement du programme suivant.

```
int main() {
2   int variable;
    pid_t idproc;
4   variable=5;

6   idproc = fork();
    if( idproc ==-1) {
8     perror("echec fork");;
      exit(EXIT_FAILURE);
10  }
    if (idproc == 0) {
12    printf("fils avant, variable = %d, idproc = %d\n", variable, idproc);
      variable=24;
14    sleep(4);
      printf("fils apres, variable = %d, idproc = %d\n", variable, idproc);
16  } else {
      printf("pere avant, variable = %d, idproc = %d\n", variable, idproc);
18    variable=12;
      sleep(10);
20    printf("pere apres, variable = %d, idproc = %d\n", variable, idproc);
      wait(NULL);
22  }
    exit(EXIT_SUCCESS);
24 }
```

3 Fork bomb

Lors de l'exécution du programme ci-dessous, combien y a-t-il de processus créés dans le cas où `N` vaut 1, 2, 3, 4, ... ? Donnez l'arborescence des processus, puis donnez la formule générale permettant de trouver le nombre de processus en fonction de N.

```
int main() {
  int i, N;
  pid_t idproc;

  printf("donner la valeur pour N\n");
  scanf ("%d", &N);
  for (i=0; i<N; i++) {
    idproc=fork();
    if (idproc==-1) {
      perror("probleme fork");
      exit (EXIT_FAILURE);
    } else {
      printf ("A- processus %d son pere est : %d\n", getpid(), getppid());
    }
    printf ("Fin processus numero %d\n", getpid());
    exit (EXIT_SUCCESS);
  }
```