Système M1 — TD 2 : Premiers processus

Semaine du 28 septembre 2009

Exercice 1 – *Exécutions à trous*

Compléter les exécutions des programmes 1, 2 et 3 (voir pages suivantes).

```
$ echo 0123456789 > toto
$ programme1
chaine lue par le processus pere:
chaine lue par le processus fils:
$ cat toto

$ programme2
Adresses de m et n dans le pere: 0xbfbcb760 0x8049808
1: Valeurs de m et n dans le pere:
Adresses de m et n dans le fils:
2: Valeurs de m et n dans le fils:
3: Valeurs de m et n dans le fils:
4: Valeurs de m et n dans le pere:
$ programme3

$ cat toto tata
```

Exercice 2 – *Calcul en parallèle*

Écrire une fonction `is_in_tab(int* tab, int length, int val)` ; qui vérifie si la valeur `val` se trouve dans `tab` de longueur `length`. La valeur de retour est une coordonnée de `val` si `tab` contient `val` et -1 sinon. On effectue une recherche en parallèle en utilisant deux processus : le père recherche dans la première partie du tableau, le fils dans la seconde. On rappelle que `WEXITSTATUS(stat)` ; permet de récupérer la valeur de retour du processus attendu par `wait(&stat)` ;.

On suspend (Ctrl-Z) le processus père avant qu'il ne récupère la valeur de son fils. Qu'affiche `ps` ?

Exercice 3 – *Foire*

Que fait le programme 4 ? Quels sont tous les affichages résultants possibles ?

Exercice 4 – *Évaluation*

```
typedef struct arit{
  char op;
  int val;
  struct arit *fgauche;
  struct arit *fdroit;
} arbre;
```

On utilise la structure `arit` pour représenter des expressions arithmétiques où :
- le champ `val` de chaque feuille contient un entier,
- le champ `op` de chaque nœud interne contient `+`, `*`, `-` ou `/`.

Écrire une fonction qui évalue une expression. Le processus évaluant un nœud créera un processus fils pour chaque sous-expression à évaluer.

Programmes

Programme 1

```
char buffer[10];
main(){
  int desc;
  desc=open("toto", O_RDWR, 0);
  if(fork()==0){
    read(desc, buffer, 2);
    printf("chaine lue par le processus fils: %s\n", buffer);
    sleep(2);
    write(desc, "ab", 2);
  } else {
    read( desc, buffer, 2);
    printf("chaine lue par le processus pere: %s\n", buffer);
    sleep(1);
    write(desc, "AB", 2);
  }
}
```

Programme 2

```
int n = 1000;
main(){
  int m=500;
  printf("Adresses de m et n dans le pere: %p %p\n",&m, &n);
  printf("1: Valeurs de m et n dans le pere: %d %d\n", m, n);
  switch(fork()){
  case (pid_t)0:
    printf("Adresses de m et n dans le fils: %p %p\n", &m, &n);
    printf("2: Valeurs de m et n dans le fils: %d %d\n", m, n);
    m *= 2; n *= 2;
    printf("3: Valeurs de m et n dans le fils: %d %d\n", m, n);
    break;
  default:
    sleep(2);
    printf("4: Valeurs de m et n dans le pere: %d %d\n", m, n);
  }
}
```

Programme 3

```
FILE* f;
main(){
  int desc = open("tata", O_WRONLY | O_CREAT | O_TRUNC, 00777);;
  f = fopen( "toto", "w");
  printf( "abcd"); fprintf( stderr, "ABCDE");
  fprintf( f, "123456"); write( desc,  "123456", 6);
  switch(fork()){
  case  (pid_t)0:
    printf("fils\n"); fprintf(stderr, "FILS\n");
    fprintf( f, "789fils\n"); write( desc,  "789fils\n", 8);
    break;
  default:
    sleep(1);
    printf("pere\n"); fprintf(stderr, "PERE\n");
    fprintf( f, "789pere\n"); write( desc,  "789pere\n", 8);
  }
}
```

Programme 4

```
void erreur(char *m) {perror(m); exit(EXIT_FAILURE);}

main(){
  int n=0;
  printf("a\n");
  while(n<=3){
    if ( (n%2 == 0) && ( fork() == -1) ) erreur("fork");
    n++;
    printf("b\n");
    if (n%2 == 1)
      switch(fork()){
      case 0:
        printf("%d\n",n);
        break;
      default:
        printf("c\n");
        exit(EXIT_SUCCESS);
      }
  }
}
```