UNIVERSITÉ PARIS SUD
IUT D'ORSAY

DUT Informatique
M3101 Système S3
2015 / 2016

Travaux Pratiques n° 1 : Processus et signaux

Nom(s) :
Groupe :
Date :

Objectifs : savoir créer les processus Unix avec l'appel système `fork()` et les synchroniser avec le processus-père par `exit()` et `wait()`. Maîtriser la communication entre processus par signaux Unix.

Exercice 1 : Aiguille dans le foin. Le but de ce problème est de détecter la présence d'un zéro dans un tableau de **`unsigned char`** de taille `TABSIZE` (par exemple, 10000) en partageant le travail entre plusieurs processus.

Nous allons utiliser le code suivant pour initialiser le tableau :

```
unsigned char arr[TABSIZE];
srandom(time(NULL));
// entasser du foin
for (i = 0; i < TABSIZE; i++)
  arr[i] = (unsigned char) (random() % 255) + 1;
// cacher l'aiguille
printf("Enter a number between 0 and %d: ", TABSIZE);
scanf(" %d", &i);
if (i >= 0 && i < TABSIZE) arr[i] = 0;
```

Consultez les pages `man 3 random` et `man 2 time` pour vous renseigner sur le fonctionnement de ces primitives (et les fichiers entêtes à inclure dans votre code source).

À quoi sert l'instruction `srandom(time(NULL))` ?

Combien de zéros, au minimum et au maximum, peuvent apparaitre dans le tableau à la fin de ce fragment de code ? Justifiez la réponse.

Une première version de votre programme doit générer un processus fils et lui confier une moitié du tableau. Le processus père devra fouiller l'autre moitié. À la fin de son travail, le fils communiquera au père le résultat de ses recherches : 1 si zéro est trouvé, sinon 0. Le père doit alors combiner les résultats et afficher le verdict final :

```
if (found) printf("Got a needle!\n");
else printf("No needles.\n");
```

De quel mécanisme peut-on se servir pour passer l'information du fils au père ?

Écrivez le programme C correspondant et **joignez le code source à votre compte rendu**.

Énumérez les bonnes valeurs d'indice à donner à votre programme pour le tester.

Exercice 2 : Des foules pour les fouilles. On modifie le programme de l'exercice précèdent pour qu'il prenne sur la ligne de commande un nombre `N` entre 1 et 100 qui désigne le nombre de processus entre lesquelles on partage le tableau pour faire la recherche :

```
if (argc > 1) N = atoi(argv[1]);
if (N < 1 || N > 100) N = 1;
```

Nous allons légèrement modifier la structure de notre algorithme. À savoir, votre programme doit créer `N` processus fils pour chercher dans le tableau, et laisser le père juste récupérer et combiner leurs résultats.

Écrivez le programme C correspondant et **joignez le code source à votre compte rendu**.

Exercice 3 : Tous dehors. Reprenons le programme de l'exercice 2. Supposons qu'un des processus fils trouve zéro très vite et en avertit le père. Dans ce cas, le père peut afficher la réponse et terminer sans attendre la fin de calcul des autres fils.

Modifiez le code de votre programme à cet effet. Pour mieux comprendre le comportement du programme, ajoutez deux commandes suivantes dans la boucle qui parcourt le tableau :

```
putc('.', stdout);
fflush(stdout);
```

Lisez les pages de `man` correspondantes et expliquez ce que font ces commandes.

Faites plusieurs tests avec le programme modifié. Que constatez-vous ?

Il nous faut un moyen d'arrêter les processus fils encore vivants dès l'instant que le père a reçu le résultat positif (et même avant de l'afficher !). Heureusement, la primitive `kill()` nous permet d'envoyer un signal à tous les processus dans le groupe de l'appelant, ce qui dans notre cas correspond à tous les processus engendrés au cours d'exécution du programme :

```
kill(0, SIGTERM);
```

Modifiez le code de votre programme à cet effet. Pour mieux comprendre le comportement du programme, faites afficher leur PID au processus père et à tous les processus fils au début de l'exécution. Outre cela, pensez à installer un gestionnaire de `SIGTERM` qui affiche le PID du processus courant et le termine par un appel de `exit()`.

Faites plusieurs tests avec le programme modifié. Que constatez-vous ? Proposez une solution au problème.

Finissez le programme et **joignez le code source à votre compte rendu**.

Exercice 4 : Mais on l'a déjà fait en TD ! Implémentez et testez l'exercice 8 du TD1. Utilisez la commande `time` pour savoir combien de temps il faut à votre machine pour compter jusqu'à 2^{32}.