TP 1 - Signaux et processus

Exercice 0 : Hello world

> Créez un fichier hello.c contenant le code ci-dessus, compilez-le et exécutez-le.

See file

Output

```
hello, world
```

Rappels pointeurs

See file

Output

```
valeur = 11.000000
&valeur = 0x7ffc2fd38ab8
```

Que représente valeur ? Qu'affiche le premier printf() ?

> `valeur` est un double. Le premier printf affiche `11.000000`

Que représente &valeur ? Qu'affiche le second printf() ?

> `&valeur` est l'addresse à laquelle se trouve la valeur de `valeur`

Que représente pv ? Que représente &pv ? Que représente *pv ?

> `pv` est un pointeur de type double qui pointe à l'adresse de valeur

Quelle est la taille de la zone mémoire réservée pour valeur ? Pour pv ? Pour nombre ? Pour pn ?

> `valeur` a une la taille d'un double, donc en C 8 octets. `pv` a la taille d'une addresse mémoire, donc 8 octets. `nombre` a la taille d'un int, donc 4/8 octets, et `pn` fait 8 octets.

Pourquoi les tailles sont-elles différentes pour valeur et nombre, mais identiques pour pv et pn ?

> cf. réponse précédente

Passage en paramètre

See file

Output

```
Avant echange : pi = 2.718280, e = 3.141590.
Apres echange : pi = 2.718280, e = 3.141590.
```

Qu'affiche ce programme ? Expliquez.

> cf. output
> le programme affiche 2 fois la même chose car les variables sont copiées et donc ne sont pas réellement échangées.

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 :

```c
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)) ?

> `srandom()` sert à générer un nombre "aléatoire" à partir d'une seed, avec toujours le même résultat.

> Ici, on veut un nombre changeant, donc on le génère avec `time(NULL)` qui nous renvoie le temps en millisecondes depuis **Epoch**, i.e. **UNIX time**.

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

> Il faut tout d'abord savoir que le tableau ne contient aucun zéro avant le *user input*.

> Si le nombre entré par l'utilisateur est une case du tableau, alors la valeur de cette case sera remplacée par un 0, puis fin de l'exécution.

> Si le nombre est négatif ou plus grand que TABSIZE, alors le programme met un 0 à la première case du tableau et fin de l'exécution.

> Ainsi, on a minimum 0 zéros et maximum 1 zéro à la fin de l'exécution.

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 ?

`WEXITSTATUS()` et `exit()` utilisées ensemble permettent de transmettre l'information du fils au père.

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

See file

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

- -1
- 0
- un nombre entre 0 et TABSIZE
- un nombre >= TABSIZE

Exercice 2 : Des foules pour des fouilles

See file

Exercice 3 : Tous dehors.

See file