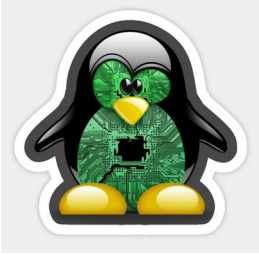



<p>L2 - « Administration système » L. Mascarilla, E. Zahzah</p>	<p>TD n°2 -</p>  <p>Compléments</p>	
--	--	---

Exercice 1. system : fork, exec, fichiers et redirections

La bibliothèque C « `stdlib.h` » fournit une fonction **`int system(const char *command);`** qui exécute une commande du shell puis permet de continuer l'exécution du programme appelant (ce qui est différent de `exec`). Par exemple :

```
#include <stdlib.h>
```

```
int main(){
    int r = system("date") ;
    printf("retour = %d\n",r) ; /* cette ligne est bien exécutée */
}
```

Affichera :

```
lun. mars  4 11:51:44 CET 2022
retour = 0
```

Vous observez que le programme se poursuit **après** l'utilisation de `system`.

1- Ecrivez un programme `systeme.c` dans lequel vous définirait une fonction **`int monsystem(char * cmd)`** ; qui simule la fonction `system` standard. Le `main()` sera similaire au précédent, en remplaçant la fonction standard par la votre.

A) Dans un premier temps, la fonction prendra comme paramètre une chaîne de caractères contenant un seul mot ("date" est valable, pas "ls -l"). Elle utilisera les primitives :

- `fork`
- `wait`
- `exec`

Si la commande est inconnue, la fonction doit afficher « **Commande inconnue** »,

Si la commande retourne un code retour il est affiché « **Code retour :** »

Si la commande est interrompue par un signal il est affiché « **Tuée par le signal :** »

Ces affichages se feront tous sur la **sortie d'erreur (stderr)**

La fonction `monsystem` doit retourner 0 si la commande s'est terminée avec le code de retour 0, et -1 dans tous les autres cas.

On rappelle que la valeur du status dans `wait(&status)` doit être utilisé ainsi :

```
if (WIFEXITED(status)) {
    // code retour dans WEXITSTATUS(status)
} else if (WIFSIGNALED(status)) {
    // numéro de signal dans WTERMSIG(status)
}
```

- Quelle variante de exec utiliserez-vous ? Pourquoi ?

B) Modifiez la fonction `monsystem` pour qu'on puisse passer une commande avec des arguments (comme `"ls -l"`).

- Quelle variante de exec utiliserez-vous ? Pourquoi ?

Remarque : Vous aurez (peut-être!) besoin de découper votre chaîne de caractères en mots. Pour cela, il est possible d'utiliser la fonction `strtok` dont le fonctionnement est illustré par ce code qui affiche chacun des mots, séparés par des espaces, de la chaîne `ch` :

```
#include <string.h>
int main ()
{
    char ch[] = "deux mots";
    char * mot[100]={0}; // 100 mots maximum
    mot[0] = strtok (ch," "); // séparateur = " "
    int i=0;
    while (mot[i] != NULL)
    {
        printf("%s\n",mot[i++]);
        mot[i] = strtok (NULL, " ");
    }

    return 0;
}
```

Attention : `strtok` modifie la chaîne de caractère traitée (ici `ch`), donc avant de l'utiliser dans une fonction pensez à effectuer une copie (par `strcpy` par exemple).

C) Si vous passez comme argument sur la ligne de commande un nom de fichier, les résultats de la commande devront être écrits dans ce fichier, mais le dernier affichage (`printf("retour = %d\n", x) ;`) sera lui écrit sur la sortie standard. Vous utiliserez pour effectuer ces redirections la primitive **dup** (ou **dup2**).

Exercice 2. Anneau : pipe (d'après Arnaud Legrand)

Le but de cet exercice est d'écrire un programme `anneau.c` qui prend en argument un nombre `n` qui désigne le nombre de processus qu'il va créer. Chacun de ces processus est relié au processus qui le précède et celui qui le suit par un tube.

Par exemple, si `n=3`, on a 3 processus (disons `A`, `B` et `C`). Le schéma est le suivant `A→B→C` et `C→A`, les

« -> » désignant un pipe.

Le processus A tire un **nombre aléatoire** et l'envoie au processus B. B tire à son tour un nombre aléatoire et envoie le plus grand des deux nombres (entre celui de A et le sien) à C. C suit le même schéma et envoie son résultat à A.

A affiche la plus grande de toutes les valeurs aléatoires.

La fonction suivante sera utilisée pour tirer un entier aléatoire :

```
int alea() {  
    srand( getpid() );  
    return rand() % 1024;  
}
```

Le programme sera structuré de la façon suivante :

Dans le programme principal (*i.e.* le père), on crée un tableau de *n* pipes puis une boucle permet de créer les *n* processus fils. Enfin, le père appelle une fonction *noeud* qui a comme paramètre son numéro d'ordre :

```
void noeud(int num) ;
```

L'appel réalisé par le père est donc *noeud(0)* ;

Chaque fils appelle la fonction *noeud(i)*, où *i* est l'indice du fils (donc compris entre 1 et *n-1*)

La fonction *noeud* se comportera (légèrement) différemment si *num==0* ou si *num>0* :

Si *num==0*, la fonction écrit son nombre aléatoire sur le premier pipe et lit la sortie du dernier pipe, qui contiendra le résultat (le maximum des nombres aléatoires). Ce résultat est affiché et le programme se termine.

Si *num>0*, la sortie du pipe précédent est lue et comparée à un nombre aléatoire puis le maximum de ces deux nombres est envoyé sur l'entrée du pipe connecté au processus suivant.