TRAVAUX PRATIQUE DE PROGRAMMATION SYSTEME
LES PROCESSUS

Conservez tous les programmes, ils évolueront au fur et à mesure des TP

PARTIE 1

Exercice 1 :

Écrire la fonction

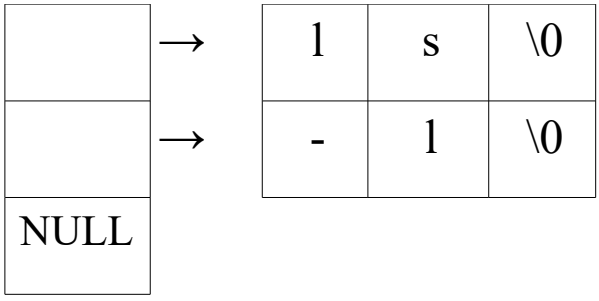
char **Ligne2Argv(char*) ;

qui transforme une chaîne de caractères contenant une commande en un tableau d'arguments de type argv (voir premier cours)

Exemple :

```
commande=Ligne2Argv("ls -l");
```

commande



Exercice 2 :

Écrire la fonction

void AfficheArgv(char) ;**

qui affiche un tableau de type ARGV.

Exemple :

```
AfficheArgv(commande) ;

Arg 0 : ls
Arg 1 : -l
```

Exercice 3:

Écrire la fonction

char *Argv2Ligne(char Commande)**

qui transforme un tableau de type ARGV en chaîne de caractères.

Exemple :

```
ligne=Argv2Ligne(commande) ;
```

ligne

l	s		-	l	\0

Exercice 4 :

Ecrire la fonction

int Executer(char **)

qui reçoit un argument de type ARGV, l'exécute, attend qu'il se termine et retourne :

- sa valeur de retour (EXITSTATUS) ou bien
- 255 si le fork n'a pas marché (pb système)
- 254 si l'exec n'a pas marché (command not found).

Exemple :

```
retour = Execute(commande) ;
```

exécutera "ls -l" et retournera 0 car la commande se sera bien passée.

Exercice 5:

Écrire un programme interpréteur de langage de commande (**MiniBash**), qui :

- Affiche un prompt : `Entrer Commandes >`
- Lit une ligne contenant une commande au clavier (fgets), l'analyse (construit Argv) et l'exécute en attendant qu'elle se termine (Execute)
- Puis affiche à nouveau un prompt, lit la ligne suivante, l'exécute, etc. tant qu'il ne rencontre pas la fin de fichier (<ctrl D>).

Exemple :

```
>MiniBash

Entrer Commande>ls
exo1.c exo1
Entrer Commande>date
```

```
Thu Oct  7 10:08:55 CEST 2021
Entrer Commande><ctrlD>
FIN
>
```

Exercice 6:

Écrire la fonction

char * File2TabArgv(char*, int *) ;**

qui lit des commandes dans un fichier, les transforme en Argv et les ranges toutes dans un tableau qu'il retourne. Il retourne en plus dans le 2ème argument le nombre de commandes lues.

Exemple :

```
char ***Tabatgv ;
int NbCommandes ;
 tabArgv=File2TabArgv("data",&nbcommandes) ;
```

Exercice 7:

Écrire le programme **ExecFile** qui prends en arguments un nom de fichier contenant des commandes, les stocke dans un tableau (File2TabArgv) puis les exécute une à une et affiche FIN une fois que toutes les commandes sont exécutées.

Exemple :

```
 > ExecFile data
```

Aide : la commande unix "sleep s" est une commande qui ne fait rien mais attend s secondes avant de se terminer. Elle vous sera utile pour verifier vos programmes.

Exemple :
si data contient les lignes :

```
sleep 1
sleep 1
sleep 10
sleep 20
```

Le programme **ExecFile** devrait attendre 32 (1+1+10+20) secondes avant d'afficher FIN.

Exercice 8:

Écrire la fonction

int ExecuteBatch(char **)

qui reçoit un argument de type ARGV, l'exécute, mais n'attend qu'il se termine

Exemple :

```
commande=Ligne2Argv("sleep 10" ) ;
retour = ExecuteBatch(commande) ;
```

n'attendra pas le 10 secondes.

Exercice 9:

Écrire le programme **ExecFileBatch** qui prends en arguments un nom de fichier contenant des commandes, les stocke dans un tableau (File2TabArgv), les exécute toutes en même temps et affiche FIN une fois que toutes les commandes sont exécutées.

Exemple : pour le fichier data, il devrait attendre 20 secondes avant d'afficher FIN.

PARTIE 2

On va maintenant manipuler une structure contenant pour chaque commande :

- son pid : numéro de processus dans lequel s'exécute la commande,
- son statut :
  - -1 pas encore exécuté,
  - 0 terminé,
  - 1 en exécution.
- son retour (EXITSTATUS), même convention que dans Execute.
- l'epoch à laquelle la commande a été lancée,
- l'epoch à laquelle la commande s'est terminée,
- son tableau Argv.

```
typedef enrc
{
 int pid ;
 int statut ;
 int retour ;
 time_t debut ;
 time_t fin ;
  char ** argv ;
} ENRCOMM ;
ENRCOMM *TabCom ;
```

Epoch : la gestion du temps en informatique se fait généralement à l'aide d'une valeur (nommée Epoch). Elle représente le nombre de secondes écoulées depuis le 1er janvier 1970. Son type de time_t sous UNIX mais c'est en fait un entier.

Pour obtenir le temps actuel il faut utiliser la fonction time(time_t *) qui retourne l'Epoch de moment de l'appel. Elle est déclarée dans le fichier time.h.

```
#include <stdio.h>
#include <time.h>
int main()
{
time_t now ;
```

```
 time(&now) ;
 printf(" l\'epoch actuel est : %d\n",now) ;
}
```

D'autres fonctions existent pour calculer à partir d'Epoch, le jour, le mois, etc.. Par exemple, strftime copiera dans une chaîne de caractères une date plus lisible. (man 2 strftime)

Exemple :

```
struct tm  ts;
char       buf[80];
     ts = *localtime(&now);
          strftime(buf, sizeof(buf), "%a %Y-%m-%d %H:%M:%S %Z", &ts);
```

Exercice 10 :

Modifier **File2TabArgv** en **File2TabCom** qui remplira TabCom à partir des lignes du fichier passé en paramètres.

```
ENRCOMM *TabCom ;
int Nbcomm ;
     TabCom= File2TabCom(argv[1], &Nbcomm) ;
```

Exercice 11 :

Modifier le programme **ExecFile** pour qu'à la fin du programme, il affiche FIN et un rapport d'exécution de toutes les commandes exécutées, ainsi que le temps total.

- la commande,
- le pid, leur retour,
- l'epoch de début,
- de fin et
- le temps d'exécution.
-

Pour le ficher d'exemple, on devrait avoir :
FIN
sleep 1 : 125 0 1633599916 1633599917 1
sleep 1 : 126 0 1633599917 1633599918 1
sleep 10 : 127 0 1633599918 1633599928 10
sleep 20 : 128 0 1633599927 1633599948 20
temps total : 32

Exercice 12 :

Modifier le programme **ExecFileBatch** pour qu'à la fin chaque commande, il affiche un rapport d'exécution dès qu'une commande vient de se terminer, puis à la fin de toutes les commandes il affiche FIN et un rapport complet sur toutes les commandes, ainsi que le temps total.
Dans l'exemple on aurait :
une seconde d'attente puis :
sleep 1 : 125 0 1633599916 1633599917 1
sleep 1 : 126 0 1633599916 1633599917 1

puis 9 secondes d'attente puis
sleep 10 : 127 0 1633599916 1633599926 10
puis 10 secondes d'attente puis
sleep 20 : 128 0 1633599916 1633599936 20
FIN
sleep 1 : 125 0 1633599916 1633599917 1
sleep 1 : 126 0 1633599916 1633599917 1
sleep 10 : 127 0 1633599916 1633599926 10
sleep 10 : 128 0 1633599916 1633599936 20
temps total : 20

Exercice 13 :

Le nombre de commandes est trop important pour que toutes soient supportées par la machine en même temps. On va donc modifier le programme **ExecFileBatch** en **ExecFileBatchLimite** pour que le nombre de processus lancés en même temps soit limité à un nombre N donné en paramètre du programme. Dans ce cas, seulement N processus seront lancés en même temps. Bien entendu, il faut occuper toute la ressource et ne pas attendre la fin des N premières commandes pour lancer les N suivantes ; une nouvelle commande sera exécutée aussitôt qu'une commande est terminée.
En plus des rapports de fin de commandes, le programme affichera le début de lancement des commandes

Exemple avec 3 :
> ExecFileBatch data 3
lancement de sleep 1 a 1633599916
lancement de sleep 1 a 1633599916
lancement de sleep 10 a 1633599916
attente de 1 seconde
sleep 1 : 125 0 1633599916 1633599917 1
sleep 1 : 126 0 1633599916 1633599917 1
lancement de sleep 20 a 1633599917
attente de 9 secondes d'attente puis
sleep 10 : 127 0 1633599916 1633599926 10
attente de 11 secondes
sleep 20 : 128 0 1633599917 1633599937 20
FIN
sleep 1 : 125 0 1633599916 1633599917 1
sleep 1 : 126 0 1633599916 1633599917 1
sleep 10 : 127 0 1633599916 1633599926 10
sleep 20 : 128 0 1633599917 1633599937 20
temps total : 21