

LAB2 - Fonctions générales



Table des matières

1 - Objectifs	2
2 - Rappels	2
3 - Retour sur le LAB1	2
4 - Accès à l'environnement	2
4.1 - Introduction.....	2
4.2 - Les paramètres	3
4.3 - Les variables d'environnement.....	4
5 - Lancement d'une commande Shell	5
6 - Gestion des erreurs	6

By Remy



1 Objectifs

Ce « LAB » présente les notions fondamentales de programmation système. Il est basé sur la mise en parallèle de notions relatives aux commandes Shell (et Scripts Shell) avec leur équivalent en langage C.

la

- Documents : Support « Linux – Les commandes », « Linux – Le script Shell » et « Linux - Vi ».
Fiche : « Rappel compilation ».
- Notions abordées : l’environnement de développement, les variables d’environnement systèmes, le débogage.
- Commandes et fichiers exploités : GCC, commandes de gestion de fichiers, DDD.
- Travail à rendre : Vous devrez répondre directement à plusieurs questions au sein de ce document. Vous le copierez sur Moodle sous le nom « LAB2_nom.pdf », vous devrez aussi rendre les différents fichiers bash (1 fichier) et C (2 fichiers), **soit en tout 4 fichiers à déposer sur Moodle**.

2 Rappels

Les principes abordés dans ce document sont conformes aux préconisations « Standard Unix Specification V4 (SUSV4) ». Les distributions Linux, à travers la standardisation « Linux Standard Base (LSB) » sont conformes à SUSV4.

Pour vous refamiliariser avec les scripts Shell, lisez et effectuez les commandes décrites dans le support « Linux – le script Shell.pdf ».

3 Retour sur le LAB1

Le LAB1 a été l'occasion de rappeler les bases du débogage. Un de vos camarades à rendu un programme bogué (il a donc eu 0, car ce dernier ne compile même pas !).

Débuguez ce programme et expliquer la ou les erreurs (**Rendez le code C corrigé sur Moodle dans un fichier appelé part-3.c**) :

Explication complémentaires ici :

```
sentence2words-failed.c
srikanth@srikanth-VirtualBox:~/Downloads$ gcc sentence2word-failed.c
gcc: error: sentence2word-failed.c: No such file or directory
gcc: fatal error: no input files
compilation terminated.
srikanth@srikanth-VirtualBox:~/Downloads$ gcc -o sentence2words-failed sentence2words-failed.c
sentence2words-failed.c: In function 'afficheTableau':
sentence2words-failed.c:13:13: warning: format '%s' expects argument of type 'char *', but argument 2 has type 'int' [-Wformat=]
13 | printf("%s\n", sentence[i][j]);
    |                ^~~~~~
    |                |
    |             char *      int
    |             %d
sentence2words-failed.c:3:1: note: include '<stdlib.h>' or provide a declaration of 'malloc'
3 | #include <stdio.h>
  | ^~~~~~
```

Au dedut mettre %d pour afficher le char

Inclure la bibliotheque stdlib.h, le code comportait une mauivaise accolade

```
sentence2words-failed.c:60:1: error: expected identifier or '(' before 'return'
60 | return 0;
    | ^~~~~~
```

```

GNU nano 4.8                                     sentence2words-failed
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void affichetableau(char ** sentence, int nbmot, int max)
{
    int i,j;

    for(i=0;i<nbmot;i++)
    {
        for(j=0;j<max;j++)
        {
            printf("%s\n", sentence[i][j]);
        }
    }
}

```

J'ai enlever le for pour le j car on nous demande d'afficher un tableau et pas une matrice donc le code se simplifie :

```

GNU nano 4.8                                     sentence2words-failed.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void affichetableau(char ** sentence, int nbmot, int max)
{
    int i,j;

    for(i=0;i<nbmot;i++)
    {

        printf("%s\n", sentence[i]);

    }
}

```

Et ajouter le fgets a la place du scanf car le scanf s'arrete quand il y a des espaces alors avec le fgets on recupere la phrase avec les espaces.

```
srikanth@srikanth-VirtualBox: ~/Documents
GNU nano 4.8 sentence2words-failed.c
}

int main(void)
{
char **words;
char sentence[150];
int i=0,j=0,k=0, max=0, word=1;
printf("Entrez votre phrase:");
fgets(sentence,150,stdin);
for(i=0;i<=strlen(sentence);i++)
{
    if(sentence[i]==' ')
    {
        word++;
        if(k>max)
        {
            max=k;
        }
    }
}
```

```
srikanth@srikanth-VirtualBox:~/Downloads$ gcc sentence2words-failed.c
srikanth@srikanth-VirtualBox:~/Downloads$ ./a.out
Entrez votre phrase:srikanth
Segmentation fault (core dumped)
srikanth@srikanth-VirtualBox:~/Downloads$
```

Aussi resoudre le probleme du core dump, on affectant a la variable word =1 pour enlever le segmentation fault

```
srikanth@srikanth-VirtualBox: ~/Downloads
GNU nano 4.8 sentence2words-failed.c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void affichetableau(char ** sentence, int nbmot, int max)
{
    int i,j;
    for(i=0;i<nbmot;i++)
    {
        printf("%d\n", sentence[i][j]);
    }
}

int main(void)
{
char **words;
char sentence[150];
int i=0,j=0,k=0, max=0, word=1;
```

Après avoir résolu les problèmes liés au core dump on obtient le résultat :

```
srikanth@srikanth-VirtualBox: ~/Documents
srikanth@srikanth-VirtualBox:~$ cd Documents/
srikanth@srikanth-VirtualBox:~/Documents$ nano sentence2words-failed.c
srikanth@srikanth-VirtualBox:~/Documents$ gcc sentence2words-failed.c
srikanth@srikanth-VirtualBox:~/Documents$ ./a.out
Entrez votre phrase:hello world
hello
world

srikanth@srikanth-VirtualBox:~/Documents$
```



(le fichier a été renommé en part-3.c mais pendant le tp le fichier était compilé sous sentence2words-failed.c)

4 Accès à l'environnement

4.1 Introduction

Il est possible d'indiquer à un programme des éléments en paramètres qu'il pourra recevoir et utiliser à sa convenance. Ces éléments peuvent être de différents types (chaîne de caractère, valeurs numériques, fichiers, etc.).

Il est aussi possible de définir des variables au niveau de l'interpréteur de commandes (Shell). Un certain nombre de ces variables sont exposées et deviennent accessibles par l'ensemble des processus descendants du processus où elles ont été déclarées (en général le Shell de login).

Tous ces éléments constituent **l'environnement d'un programme**.

4.2 Les paramètres

En Shell :

En vous appuyant sur le document « Support Linux – Le script Shell », réalisez un script qui effectue le traitement suivant :

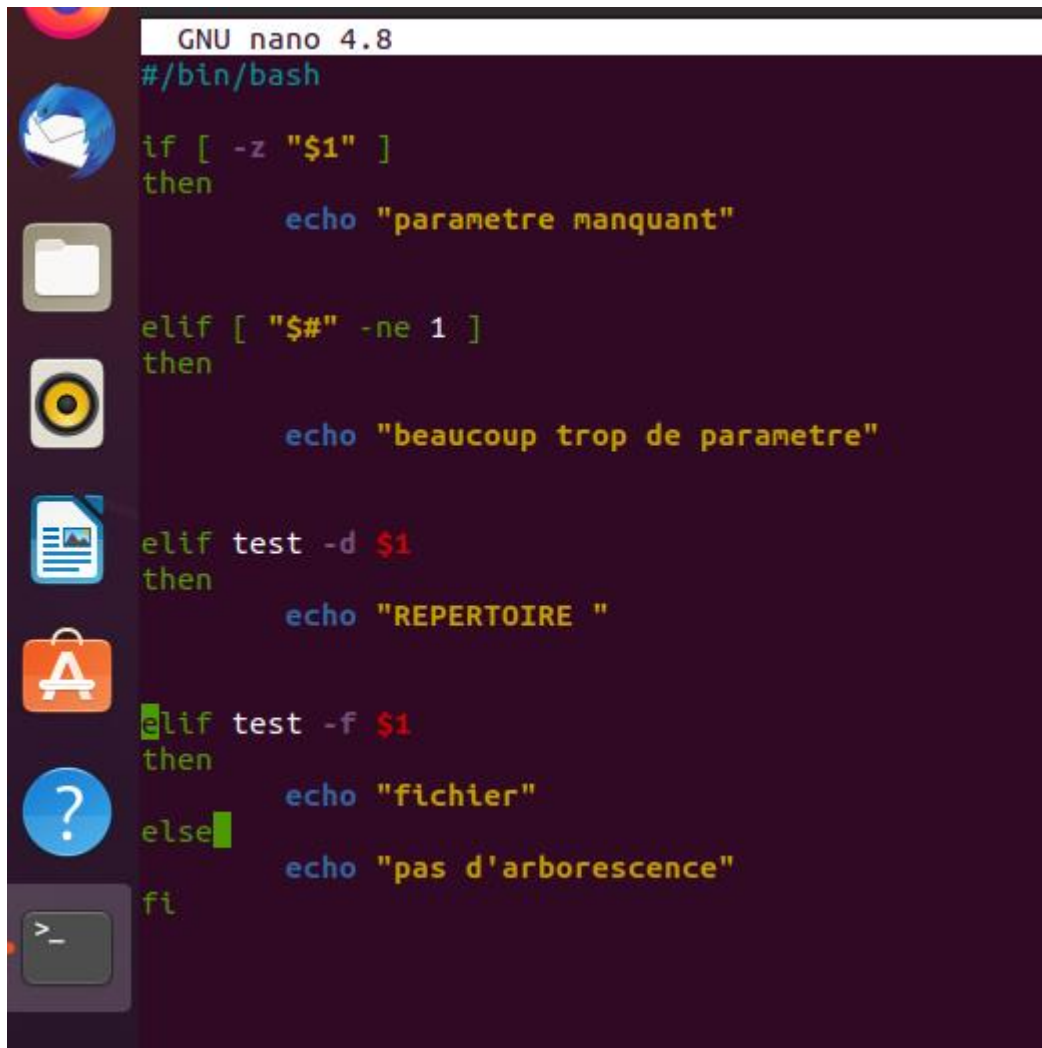
- testez le paramètre que vous passez en argument ;
- si le paramètre est un répertoire : affiche "XXX : est un répertoire" ;
- si le paramètre est un fichier ordinaire : affiche "XXX : est un fichier" ;
- sinon affiche un message d'erreur (voir figure suivante).

le
sur
dans

part-

```
[rexy@localhost ~]$ ./script2.sh tmp
    tmp est un répertoire.
[rexy@localhost ~]$ ./script2.sh script1.sh
    script1.sh est un fichier ordinaire
[rexy@localhost ~]$ ./script2.sh foo
    foo n'existe pas dans l'arborescence ou est d'un autre type.
[rexy@localhost ~]$ ./script2.sh
    Le paramètre est manquant
[rexy@localhost ~]$ ./script2.sh foo bar
    Trop de paramètres
[rexy@localhost ~]$
```

Rendez
script
Moodle
un
fichier
appelé
4.sh



```
GNU nano 4.8
#!/bin/bash

if [ -z "$1" ]
then
    echo "parametre manquant"

elif [ "$#" -ne 1 ]
then
    echo "beaucoup trop de parametre"

elif test -d $1
then
    echo "REPERTOIRE "

elif test -f $1
then
    echo "fichier"
else
    echo "pas d'arborescence"
fi
```

En C :

main() est une fonction comme une autre. Elle reçoit des paramètres et renvoie une valeur.

```
int main (int argc, char *argv[], char *envp[]) ;
```

Explication des paramètres :

- « int argc » (argument count) : nombre d'arguments passés au programme ; y compris le nom du programme lui-même. Ainsi, cette variable ne peut jamais être nulle ;
- « char *argv[] » (argument value) : pointeur vers un tableau de pointeurs pointant vers un tableau de caractère de format chaîne (dernier caractère = 0). Chaque chaîne contient le nom d'un argument passé au programme. Argv[0] contient le nom du programme lui-même. Le tableau se termine (argv[argc]) par un pointeur de valeur nulle ;
- « char envp[] » (environment program) : comme pour argv, mais le tableau de chaîne contient les variables d'environnement exportées par les processus ascendants. La norme SUSV4 préconise de ne plus utiliser ce pointeur au profit d'un autre dispositif (cf. § suivant) ;

- Valeur renvoyée (int) : entier utilisable par le processus appelant (en Shell, elle est récupérée par \$?).

Exemple :

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    printf ("Je me nomme %s et on m'a envoyé %d paramètres\n",
argv[0],argc-1);
}
```

Copiez ici une capture d'écran de son exécution avec 3 paramètres.

```
srikanth@srikanth-VirtualBox:~/Downloads$ nano tp2.c
srikanth@srikanth-VirtualBox:~/Downloads$ gcc -o tp2 tp2.c
srikanth@srikanth-VirtualBox:~/Downloads$ ./tp2
je me nomme ./tp2 et on m'a envoyer 0 parametres
srikanth@srikanth-VirtualBox:~/Downloads$

srikanth@srikanth-VirtualBox:~/Downloads$ cat tp2.c
#include <stdio.h>

int main(int argc, char *argv[],char *envp[])
{
printf("je me nomme %s et on m'a envoyer %d parametres\n" ,argv[0],argc-1);
}
srikanth@srikanth-VirtualBox:~/Downloads$
```

4.3 Les variables d'environnement

En Shell :

Quelles commandes permettent de connaître les variables d'environnement définies sur votre système ?

La commande qui permettent de connaître les variables d'environnement définies sur le système et la commande printenv


```

srikanth@srikanth-VirtualBox:~/Downloads$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/srikanth-VirtualBox:@/tmp/.ICE-unix/1759,unix/srikanth-VirtualBox:/tmp/.ICE-unix/1759
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LC_ADDRESS=fr_FR.UTF-8
GNOME_SHELL_SESSION_MODE=ubuntu
LC_NAME=fr_FR.UTF-8
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
LC_MONETARY=fr_FR.UTF-8
SSH_AGENT_PID=1710
GTK_MODULES=gail:atk-bridge
PWD=/home/srikanth/Downloads
LOGNAME=srikanth
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/srikanth
USERNAME=srikanth
IM_CONFIG_PHASE=1
LC_PAPER=fr_FR.UTF-8
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:s
g=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha
=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz
=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz
=01;31:*.tbz2=01;31:*.taz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.al
z=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.e
sd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;
35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx
=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.
mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35
:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35
:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00
;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
XDG_CURRENT_DESKTOP=ubuntu:GNOME
VTE_VERSION=6003
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/07d61441_62b4_400a_9bfd_1e87718fc243
INVOCATION_ID=6d2419c64f944082a7b2318380ff4a33
MANAGERPID=1535
GJS_DEBUG_OUTPUT=stderr
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LC_IDENTIFICATION=fr_FR.UTF-8
LESSOPEN=| /usr/bin/lesspipe %s
USER=srikanth
GNOME_TERMINAL_SERVICE=:1.97
DISPLAY=:0
SHLVL=1
LC_TELEPHONE=fr_FR.UTF-8
QT_IM_MODULE=ibus
LC_MEASUREMENT=fr_FR.UTF-8
XDG_RUNTIME_DIR=/run/user/1000
LC_TIME=fr_FR.UTF-8
JOURNAL_STREAM=8:35032
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
LC_NUMERIC=fr_FR.UTF-8
OLDPWD=/home/srikanth
_=/usr/bin/printenv
srikanth@srikanth-VirtualBox:~/Downloads$

```

- Créez une nouvelle variable « école » initialisée à « esiea » et affichez-la.
- Créez un Script-Shell qui affiche cette variable.
- Que devez-vous faire pour que cette variable soit connue de votre script ?

Détail des commandes et du script :

Voici les étapes pour créer une variable ecole et initialisees a esiea


```
srikanth@srikanth-VirtualBox:~/Downloads$ export VAR="ECOLE"
srikanth@srikanth-VirtualBox:~/Downloads$ echo $VAR = "ESIEA"
ECOLE = ESIEA
srikanth@srikanth-VirtualBox:~/Downloads$
```

En C :

La primitive « **getenv()** » permet d'obtenir la valeur courante d'une variable d'environnement particulière (**man 3 getenv**).

```
#include <stdlib.h>
char* getenv (char *string) ;
```

Explication des paramètres :

- paramètre « char *string » : chaîne contenant la variable dont on veut obtenir la valeur ;
- valeur renvoyée (char*) : pointeur sur une zone mémoire statique contenant la chaîne correspondant à la variable demandée si celle-ci existe.

Exemple :

```
#include <stdlib.h>
#include <stdio.h>
void main (void)
{
    char *ptr ; /* récupère le résultat de « getenv() » */

    ptr=getenv("PATH") ;
    printf("PATH = %s\n", ptr) ;
}
```

Copiez ici une capture d'écran de son exécution.

```
srikanth@srikanth-VirtualBox:~/Downloads$ nano tp22.c
srikanth@srikanth-VirtualBox:~/Downloads$ gcc -o tp22 tp22.c
srikanth@srikanth-VirtualBox:~/Downloads$ ./tp22
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
srikanth@srikanth-VirtualBox:~/Downloads$
```

Toutes ces variables d'environnement héritées sont présentes dans un tableau de chaînes **extern char **environ** qu'il faut déclarer avant exploitation (**man 7 environ**).

Exemple :

```
#include <stdio.h>
extern char **environ ;
int main (void)
{
    int nro_var=0 ;
    for (nro_var=0 ; environ[nro_var]!= NULL ; nro_var++)
```

```
printf ("%d : \t%s\n", nro_var, environ[nro_var]) ;  
return 0 ;  
}
```

5 Lancement d'une commande Shell

En Shell :

Rappel - Le « | » permet de chaîner les commandes Shell (la sortie standard de la première commande est fournie en entrée de la deuxième commande et ainsi de suite).

Copiez ici la suite de commandes Shell permettant de connaître le nombre de variables d'environnement déclarées sur votre système ? Quel est ce nombre ?

```
srikanth@srikanth-VirtualBox:~$ printenv | wc -l  
58  
srikanth@srikanth-VirtualBox:~$
```

Il y a 58 variables.

En C :

Il est possible, depuis un programme C, de lancer l'exécution d'une commande Shell via l'appel de la fonction « **system()** » (man 3 system). Cette fonction lance un Shell qui va exécuter la commande voulue. Le Shell lancé ne réalise aucune communication avec le programme appelant. Celui-ci reste en attente jusqu'à la fin de l'exécution de la commande.

Remarque : Il est parfois plus judicieux de reprogrammer en C la commande que l'on veut faire exécuter. En effet, la rapidité d'exécution est accrue et la communication est possible.

```
#include <stdlib.h>  
int system (char *commande) ;
```

Explication des paramètres :

- paramètre « char *commande » : chaîne contenant la commande que l'on veut exécuter ;
- Valeur renvoyée (int) : un nombre contenant le statut de fin du Shell lancé.

Exemple :

```
#include <stdlib.h>  
main ()  
{  
    system ("date") ;  
}
```

Réalisez un programme C qui appelle une commande Shell (ou une suite de commandes « pipées ») permettant d'afficher le nombre de variables d'environnement de votre système. Le programme doit aussi afficher le statut de retour d'appel de la commande shell, collez une capture d'écran de son exécution ci-dessous (**Rendez le code C sur Moodle dans un fichier appelé part-5.c**).

```
GNU nano 4.8 part
#include<stdlib.h>
int main()
{
system("printenv | wc -l");
}

srikanth@srikanth-VirtualBox:~/Downloads$ nano part-5.c
srikanth@srikanth-VirtualBox:~/Downloads$ gcc part-5.c
srikanth@srikanth-VirtualBox:~/Downloads$ ./a.out
58
```

6 Gestion des erreurs

Le traitement des erreurs et leur analyse permettent d'améliorer la stabilité du programme tout en assurant un bon niveau de maintenabilité. Pour ces deux raisons, il est nécessaire de savoir gérer les erreurs et exploiter le débogueur.

En Shell :

Un script Shell s'exécute séquentiellement. Quand il rencontre une erreur : il s'arrête ; il identifie l'erreur ; il affiche un message sur le canal 2 des erreurs (ce canal est dirigé par défaut vers l'écran) et il continue la lecture du script. Il est possible de rediriger ce canal vers un fichier en mode concaténation (2>>error_file) ou vers un fichier vide (2>/dev/null).

Exemple :

```
#!/bin/bash
NB_USERS=`cat /etc/passwd | wc -l`    # le fichier "/etc/passwd" n'existe
pas (c'est "/etc/passwd")
if [ $NB_USERS -gt 0 ]
then
    echo "$NB_USERS sont déclarés sur votre système"
else
    echo "une erreur a dû se produire"
fi
```

En termes de débogage, le développeur peut afficher l'état de sortie de chaque commande de son script afin de suivre précisément son exécution (cf. §2.3 du document « support Linux – le script Shell »).

Copiez ici le résultat d'exécution de ce script avec et sans redirection des erreurs. Testez l'option de suivi d'exécution des commandes.

```

srikanth@srikanth-VirtualBox: ~/Downloads$ sh -x tp23.sh
+ wc -l
+ cat /etc/passwd
cat: /etc/passwd: No such file or directory
+ NB_USERS=0
+ [ 0 -gt 0 ]
+ echo une erreur a du se produire
une erreur a du se produire
srikanth@srikanth-VirtualBox:~/Downloads$

```

En C :

Par défaut, les appels système en échec renvoient « -1 » ou parfois « NULL ». Pour fournir un complément d'information au programme appelant, la variable « **extern int errno** » contient un numéro d'erreur indiquant la cause réelle de l'échec (**man 3 errno**). À chaque numéro correspond une constante prédéfinie. Cette variable et les constantes associées sont déclarées dans le fichier header « **errno.h** ».

```

#define EPERM      1      /* Operation not permitted */
#define ENOENT     2      /* No such file or directory */
#define ESRCH      3      /* No such process */
#define EINTR      4      /* Interrupted system call */
#define EIO        5      /* I/O error */
#define ENXIO      6      /* No such device or address */
etc.

```

Associée à cette variable, un pointeur sur un tableau de chaînes statiques « **extern char *strerror (int _ernum)** » peut être exploitée pour connaître le descriptif de l'erreur en fonction de son numéro (**man 3 strerror**). Ce pointeur est déclaré dans le fichier header « **string.h** ».

Exemple :

```

#include <stdio.h>
#include <errno.h>
#include <string.h>
void main (void)
{
    FILE *file_desc = NULL ;
    if ((file_desc = fopen("/etc/password", "r") == NULL) //le fichier
"/etc/password" n'existe pas (c'est "/etc/passwd")
    {
        printf ("Erreur numéro : %d\n", errno) ;
        printf ("Description : %s\n", strerror(errno)) ;
    }
    else fclose (file_desc) ;
}

```

Une solution plus directe consiste à exploiter la fonction **void perror (const char *s)** déclarée dans le fichier header « **stdio.h** ». Cette fonction affiche **sur la sortie des erreurs** un texte choisi par le

programmeur immédiatement suivi du message d'erreur correspondant à **errno**.

Exemple

```
#include <stdio.h>
void main (void)
{
FILE *file_desc = NULL ;
    if ((file_desc = fopen("/etc/password", "r")) == NULL)    //le fichier
"/etc/password" n'existe pas (c'est "/etc/passwd")
    {
        perror ("Erreur d'ouverture de fichier ") ;
    }
    else fclose (file_desc) ;
}
```

- Testez le programme ci-dessus. **Conseil :** les bonnes pratiques de programmation exigent **de tester systématiquement** la valeur de retour liée au résultat d'exécution des fonctions.
- En utilisant la variable externe « environ » (cf. man), écrivez un programme C qui calcule et affiche le nombre de variables système. Exploitez le débogueur DDD pour lancer votre programme en mode « pas à pas » et afficher la variable « environ[nro_variable] » dans le cadre de suivi des variables.

Faites une copie d'écran de ce débogueur quand il est sur la 29e variable système. Le débogueur doit montrer le code source et le contenu de la variable environ[nro_variable].