

## UE Programmation Unix

## TP2 Gestion des Processus

## Les appels système : gestion de processus (suite)



**Un appel système** permet à un processus utilisateur d'accéder à une ou plusieurs fonctions internes au noyau du système d'exploitation et de les exécuter.

Ainsi, un programme peut invoquer un ou plusieurs services du système d'exploitation.



## Concept de Processus

Définition **PROCESSUS** – un programme en cours d'exécution

Un processus est plus que le code d'un programme, qui est parfois appelé **la section de texte**.

- Un programme par lui-même n'est pas un processus : un programme est une **entité passive**, tel que le contenu d'un fichier stocké sur disque, tandis qu'un processus est une **entité active**.
- Bien que deux processus puissent être associés au même programme, ils sont toutefois considérés comme deux séquences d'instructions séparées.  
Par exemple, plusieurs utilisateurs peuvent exécuter plusieurs copies du programme de gestion de courrier électronique ou le même utilisateur peut appeler plusieurs copies du programme éditeur.
- Chacune d'elles est un processus séparé et bien que **les sections de texte** soient équivalentes, les sections de données ne sont pas les mêmes.



# La fonction main()

## Arguments de la ligne de commandes

- Langage C offre des mécanismes qui permettent d'intégrer parfaitement un programme C dans l'environnement hôte
  - environnement orienté ligne de commande (Unix, Linux)
- Programme C peut recevoir de la part de l'interpréteur de commandes qui a lancé son exécution, une liste d'arguments
  - ⇔ ligne de commande qui a servi à lancer l'exécution du programme
- Liste composée
  - du nom du fichier binaire contenant le code exécutable du programme
  - des paramètres de la commande

## La fonction main ()

Un processus débute par l'exécution de la fonction `main()` du programme correspondant

### Definition

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

- `argc`: nombre d'arguments de la ligne de commande y compris le nom du programme
- `argv[]`: tableau de pointeurs vers les arguments (paramètres) passés au programme lors de son lancement

### A NOTER

- `argv[0]` pointe vers le nom du programme
- `argv[argc]` vaut `NULL`

- `argc` (argument count)
  - nombre de mots qui compose la ligne de commande (y compris le nom de la commande qui a servi à lancer l'exécution du programme)
- `argv` (argument vector)
  - tableau de chaînes de caractères contenant chacune un mot de la ligne de commande
  - `argv[0]` est le nom du programme exécutable

## Exemple de cours

```
#include <stdio.h>

int main (int argc, char *argv[]){
    int i;
    for (i=0;i<argc;i++){
        printf ("argv[%d]: %s\n",i, argv[i]);
    }
    return (0);
}

>./affich_param Bonjour à tous
argv[0]: ./affich_param
argv[1]: Bonjour
argv[2]: à
argv[3]: tous
```

```
[ij04115@saphyr:~/unix_tpl]:ven. sept. 11$ nano affich_param.c
```

saphyr.ens.math-info.univ-paris5.fr - PuTTY

GNU nano 2.5.3 Fichier : affich\_param.c

```
#include <stdio.h>

int main(int argc, char *argv[]){
    int i;
    for (i = 0 ; i < argc ; i++){
        printf("argv[%d]: %s\n", i, argv[i]);
    }
    return (0);
}

} //main
```

Spécifier le nom de l'exécutable avec l'option -o

```
ProgC > gcc -o toto premierProg.c
ProgC > ls
toto    premierProg.c
ProgC > ./toto
```

```
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$ gcc -o affich_param affich_param.c
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$ ls
affich_param  affich_param.c
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$
```

```
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$ ./affich_param Bonjour à tous
argv[0]: ./affich_param
argv[1]: Bonjour
argv[2]: à
argv[3]: tous
[ij04115@saphyr:~/unix_tpl]:sam. sept. 12$
```



# Le fichier standard des erreurs **stderr**

## Structure *FILE* \* et variables *stdin*, *stdout* et *stderr*

Entête à inclure

```
#include <stdio.h>
```

Structure *FILE* \* et variables *stdin*, *stdout* et *stderr*

```
FILE * stdin;
FILE * stdout;
FILE * stderr;
```

La structure *FILE* permet de stocker les informations relatives à la gestion d'un flux de données. Néanmoins, il est très rare que vous ayez besoin d'accéder directement à ses attributs.

Effectivement, il existe un grand nombre de fonctions qui acceptent un paramètre basé sur cette structure pour déterminer ou contrôler divers aspects.

- **stdin (Standard input)** : ce flot correspond au flux standard d'entrée de l'application. Par défaut, ce flux est associé au clavier : vous pouvez donc acquérir facilement des données en provenance du clavier. Quelques fonctions utilisent implicitement ce flux (**scanf**, par exemple).
- **stdout (Standard output)** : c'est le flux standard de sortie de votre application. Par défaut, ce flux est associé à la console d'où l'application a été lancée. Quelques fonctions utilisent implicitement ce flux (**printf**, par exemple).
- **stderr (Standard error)** : ce dernier flux est associé à la sortie standard d'erreur de votre application. Tout comme stdout, ce flux est normalement redirigé sur la console de l'application.

**fprintf** it is the same as **printf**,  
except now you are also specifying the place to print to :

```
printf("%s", "Hello world\n"); // "Hello world" on stdout (using printf)
fprintf(stdout, "%s", "Hello world\n"); // "Hello world" on stdout (using fprintf)
fprintf(stderr, "%s", "Stack overflow!\n"); // Error message on stderr (using fprintf)
```



# Messages d'erreurs affiches avec perror

## C Library - <stdio.h>

### C library function - perror()

#### Description

The C library function **void perror(const char \*str)** prints a descriptive error message to stderr. First the string **str** is printed, followed by a colon then a space.

#### Declaration

Following is the declaration for perror() function.

```
void perror(const char *str)
```

#### Parameters

- **str** – This is the C string containing a custom message to be printed before the error message itself.

#### Return Value

This function does not return any value.

```
1 #include <stdio.h>
2
3 int main () {
4     FILE *fp;
5
6     /* first rename if there is any file */
7     rename("file.txt", "newfile.txt");
8
9     /* now let's try to open same file */
10    fp = fopen("file.txt", "r");
11    if( fp == NULL ) {
12        perror("Error: ");
13        return(-1);
14    }
15    fclose(fp);
16
17    return(0);
18 }
```

Let us compile and run the above program that will produce the following result because we are trying to open a file which does not exist –

```
Error: : No such file or directory
```

**Tous les programmes devront être développés avec passage de leurs éventuels paramètres à la fonction**

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

- ☐ Les valeurs de retour des appels aux primitives devront être testées et les messages d'erreurs affichés avec `perror`.
- ☐ Les messages d'erreurs à destination de l'utilisateur se feront sur le fichier standard des erreurs `stderr`.

# 1

## Le recouvrement Les primitives `exec( )`

### Le recouvrement de programme

#### Cas d'utilisation du `fork`

Le `fork` est utilisé pour :

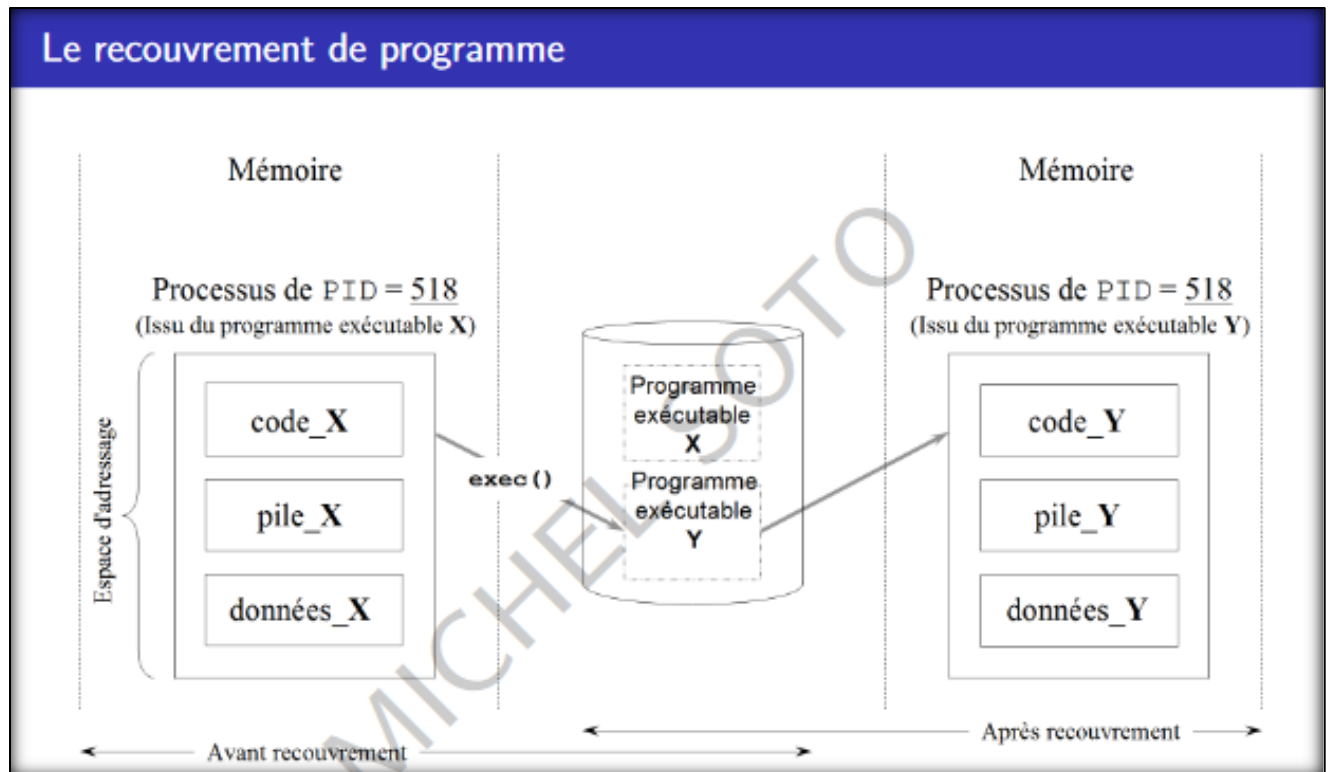
- ❶ Permettre au père et au fils d'exécuter chacun et en concurrence *une partie différente du même programme*
  - Cas d'un serveur
- ❷ Permettre au fils d'exécuter *un programme différent du père* en concurrence avec son père
  - Cas du `shell`

#### RECOUVREMENT

Dans le 2<sup>e</sup> cas, ce sont les primitives de la famille `exec` qui vont permettre au processus fils de *recouvrir (écraser) intégralement* son code exécutable avec un autre code exécutable

#### REMARQUE

Il n'est pas nécessaire d'effectuer un `fork` avant de réaliser un `exec`



Les primitives de la famille **exec ( )** permettent de **charger en mémoire de nouveaux programmes binaires en vue de leur exécution.**

Les primitives de la famille **exec ( )** se différencient par la manière dont les arguments sont transmis.

**Ces arguments sont transmis**

- ☐ Soit sous **forme d'un tableau** (famille **execv ( )**)
- ☐ Soit sous **forme de liste** (famille **exec1 ( )**)

**Selon que la primitive utilisée a un nombre fixe ou variable de paramètres.**



```
#include <unistd.h>
int execl (const char *path, const char *arg0, ... /* (char *)0 */);
int execlp (const char *file, const char *arg0, ... /* (char *)0 */);
int execl_e (const char *path, const char *arg0, ... /* (char *)0 */,
             char *const envp[]);
int execv (const char *path, char *const argv[]);
int execvp (const char *file, char *const argv[]);
int execve (const char *path, char *const argv[], char *const envp[]);
```

- Le premier argument doit pointer sur le nom du fichier associé au programme à exécuter
- Renvoient -1 en cas d'échec.

### ATTENTION

*Ces primitives ne retournent pas en cas de succès mais uniquement en cas d'échec*

- La transmission des arguments se fait soit :
  - par liste (suffixes l)
  - par un tableau de pointeur sur des chaînes de caractères (suffixes v)
- Le fichier à exécuter est soit :
  - recherché en utilisant la variable d'environnement PATH (suffixe p)
  - indiqué dans le paramètre path (absence de suffixe p)
- L'environnement peut :
  - être modifié (suffixe e)
  - être conservé (absence de suffixe e)



**Ecrire un programme qui charge un nouveau programme binaire et dont les arguments sont transmis selon les deux modes précédents.**



```

#include <unistd.h> // execl
#include <stdio.h> //fprintf, perror
#include <stdlib.h> //exit

int main(int argc, char *argv[]) {

    if(argc < 2){
        fprintf(stderr, "Utilisation: %s patch\n", argv[0]);
        exit (1);
    } //if1

    /* int execl(const char *path, const char *arg0 ...) */
    if ( execl(argv[1], argv[1], NULL) ){
        perror(argv[1]);
        exit(2);

    } //if2
} //main

```

Le second if est inutile puisque **exec** ne retourne qu'en cas d'échec

```

#include <unistd.h> // execl
#include <stdio.h> //fprintf, perror
#include <stdlib.h> //exit

int main(int argc, char *argv[]) {

    if(argc < 2){
        fprintf(stderr, "Utilisation: %s patch\n", argv[0]);
        exit (1);
    } //if

    /* int execl(const char *path, const char *arg0 ...) */
    execl(argv[1], argv[1], NULL);
    perror(argv[1]);
    exit(2);
} //main

```

In **execl** we can directly pass the pointers as arguments.  
These arguments should be NULL terminated.

```
[ij04115@saphyr:~]:sam. sept. 26$ mkdir unix_tp2
[ij04115@saphyr:~]:sam. sept. 26$ cd unix_tp2
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ nano question1_A1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ gcc question1_A1.c -o question1_A1
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ls
question1_A1  question1_A1.c
```

```
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ nano bonjour.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ cat bonjour.c
#include <stdio.h> //printf

int main() {
    printf("Hello world !\n");
    return 0;
}
```

```
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./bonjour
Hello world !
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ █
```

```
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ls
bonjour  bonjour.c  question1_A1  question1_A1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./question1_A1
Utilisation: ./question1_A1 patch
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./question1_A1 ./bonjour
Hello world !
```

```
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./question1_A1 /bin/ls
bonjour  bonjour.c  question1_A1  question1_A1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ █
```

```

#include <unistd.h> //execv
#include <stdio.h> //fprintf
#include <stdlib.h> //exit

int main(int argc, char *argv[]) {

    if(argc == 1) {
        fprintf(stderr, "Utilisation: %s command_patch [arg]\n", argv[0]);
        exit(1);
    } //if

    /* int execv (const char *path, char *const argv[]); */
    execv(argv[1], &argv[1]);
    perror(argv[1]); // only get here on error
    exit(2);

} //main

```

```

[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ nano questionl_A2.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ gcc questionl_A2.c -o questionl_A2
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A2
Utilisation: ./questionl_A2 command_patch [arg]
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A2 ./bonjour
Hello world !
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A2 /bin/ls
bonjour  bonjour.c  questionl_A1  questionl_A1.c  questionl_A2  questionl_A2.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A2 /bin/ls -l
total 36
-rwxr-xr-x 1 ij04115 licence3in 7352 sept. 26 14:33 bonjour
-rw-r--r-- 1 ij04115 licence3in   83 sept. 26 14:31 bonjour.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 14:53 questionl_A1
-rw-r--r-- 1 ij04115 licence3in  346 sept. 26 14:52 questionl_A1.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 15:06 questionl_A2
-rw-r--r-- 1 ij04115 licence3in  368 sept. 26 15:06 questionl_A2.c

```

```

[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A2 ./questionl_A1 ./bonjour
Hello world !
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A2 ./questionl_A1
Utilisation: ./questionl_A1 patch
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ █

```

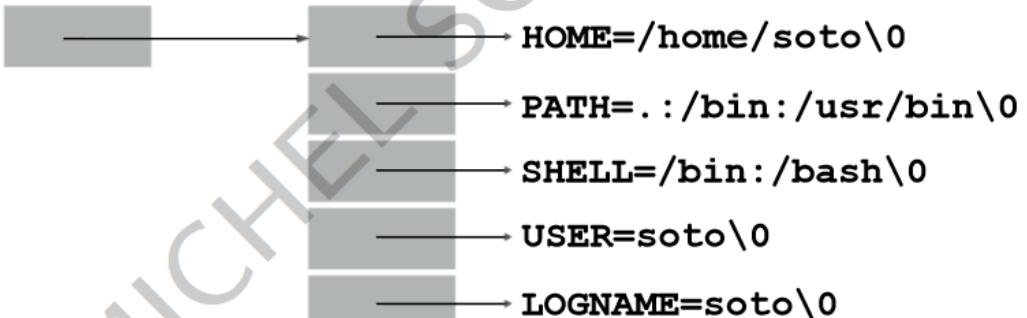
Les primitives de la famille `exec()` se différencient également par la manière dont le programme à charger est recherché dans le système de fichiers.

**Soit la recherche est relative au répertoire courant, soit elle l'est par rapport aux répertoires spécifiés via la variable `PATH`.**

```
#include <unistd.h>
int execl (const char *path, const char *arg0, ... /* (char *)0 */);
int execlp (const char *file, const char *arg0, ... /* (char *)0 */);
int execlx (const char *path, const char *arg0, ... /* (char *)0 */);
int execv (const char *path, char *const argv[]);
int execvp (const char *file, char *const argv[]);
int execve (const char *path, char *const argv[], char *const envp[]);
```

- Le fichier à exécuter est soit :
  - recherché en utilisant la variable d'environnement `PATH` (suffixe `p`)
  - indiqué dans le paramètre `path` (absence de suffixe `p`)

environ



```
HOME=/home/soto\0
PATH=./bin:/usr/bin\0
SHELL=/bin:/bash\0
USER=soto\0
LOGNAME=soto\0
```

**b** Ecrire un programme qui charge un nouveau programme binaire qui est recherché selon les deux modes précédents.

```

#include <unistd.h> // execlp
#include <stdio.h> //fprintf, perror
#include <stdlib.h> //exit

int main(int argc, char *argv[]) {
    if(argc < 2){
        fprintf(stderr, "Utilisation: %s file\n", argv[0]);
        exit (1);
    } //if

    /* int execl(const char *file, const char *arg0 ...) */
    execlp(argv[1], argv[1], NULL);
    perror(argv[1]);
    exit(2);
} //main

```

```

#include <unistd.h> //execvp
#include <stdio.h> //fprintf
#include <stdlib.h> //exit

int main(int argc, char *argv[]) {
    if(argc == 1) {
        fprintf(stderr, "Utilisation: %s command_file [arg]\n", argv[0]);
        exit(1);
    } //if

    /* int execvp (const char *file, char *const argv[]); */
    execvp(argv[1], &argv[1]);
    perror(argv[1]); // only get here on error
    exit(2);
} //main

```

```

[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ nano questionl_B1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ gcc questionl_B1.c -o questionl_B1
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ls
bonjour  bonjour.c  questionl_A1  questionl_A1.c  questionl_A2  questionl_A2.c  questionl_B1  questionl_B1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_B1
Utilisation: ./questionl_B1 file
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_B1 ./bonjour
Hello world !
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_B1 bonjour
bonjour: No such file or directory
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_B1 /bin/ls
bonjour  bonjour.c  questionl_A1  questionl_A1.c  questionl_A2  questionl_A2.c  questionl_B1  questionl_B1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_B1 ls
bonjour  bonjour.c  questionl_A1  questionl_A1.c  questionl_A2  questionl_A2.c  questionl_B1  questionl_B1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A1 /bin/ls
bonjour  bonjour.c  questionl_A1  questionl_A1.c  questionl_A2  questionl_A2.c  questionl_B1  questionl_B1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A1 ls
ls: No such file or directory
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$

```

```

[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A1 ls -l
ls: No such file or directory
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_A1 /bin/ls -l
bonjour      questionl_A1      questionl_A2      questionl_B1      questionl_B2
bonjour.c    questionl_A1.c    questionl_A2.c    questionl_B1.c    questionl_B2.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./questionl_B2 ls -l
total 60
-rwxr-xr-x 1 ij04115 licence3in 7352 sept. 26 14:33 bonjour
-rw-r--r-- 1 ij04115 licence3in   83 sept. 26 14:31 bonjour.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 14:53 questionl_A1
-rw-r--r-- 1 ij04115 licence3in  346 sept. 26 14:52 questionl_A1.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 15:06 questionl_A2
-rw-r--r-- 1 ij04115 licence3in  368 sept. 26 15:06 questionl_A2.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 15:52 questionl_B1
-rw-r--r-- 1 ij04115 licence3in  347 sept. 26 15:51 questionl_B1.c
-rwxr-xr-x 1 ij04115 licence3in 7504 sept. 26 15:58 questionl_B2
-rw-r--r-- 1 ij04115 licence3in  372 sept. 26 15:57 questionl_B2.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$

```

Les primitives de la famille **exec ( )** se différencient enfin par l'environnement conservé par le processus après recouvrement. Soit l'environnement reste inchangé, soit un nouvel environnement est transmis en paramètre.

- L'environnement peut :
  - être modifié (suffixe e)
  - être conservé (absence de suffixe e)

```
#include <unistd.h>
int execl (const char *path, const char *arg0, ... /* (char *)0 */);
int execlp (const char *file, const char *arg0, ... /* (char *)0 */);
int execle (const char *path, const char *arg0, ... /* (char *)0 */,  
            char *const envp[]);
int execv (const char *path, char *const argv[]);
int execvp (const char *file, char *const argv[]);
int execve (const char *path, char *const argv[], char *const envp[]);
```

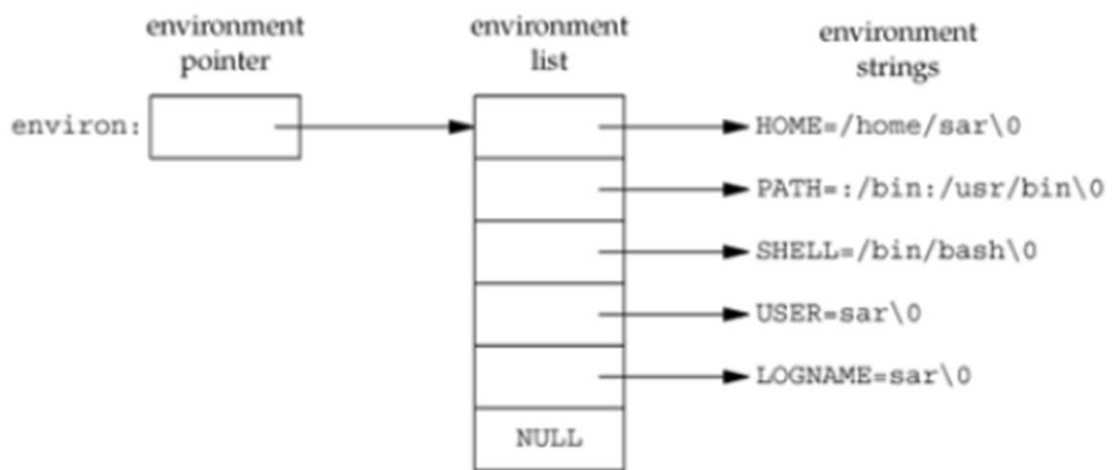
**G** Ecrire un programme qui charge un nouveau programme binaire et qui dans un premier cas conserve le même environnement et dans un second cas acquière un nouvel environnement.



- ❑ Each program also passed an environment list
- ❑ This list is an array of char pointers, with each pointer containing the address of a null-terminated C string
- ❑ The address of the array of pointers is contained in the global variable *environ*: `extern char **environ;`

### ■ Example of environment with five strings

- the null bytes at the end of each string are explicitly shown



- Historically, most UNIX systems have provided a third argument to the main function that is the address to the environment list

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

```

#include <stdio.h>

int main( int argc, char *argv[], char *envp[] )
{
    int i;
    /* echo all environment args */
    for (i = 0 ; envp[i] ; i++)
        printf( "envp[%d]: %s\n", i, envp[i] );
}
  
```

Source:

<http://www2.cs.uidaho.edu/~krings/CS270/Notes.S10/270-F10-20.pdf>

```

GNU nano 2.5.3                                Fichier : environment_list.c

#include <stdio.h>

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

    /* echo all environment args */
    for( i = 0 ; envp[i] != NULL ; i++ )
        printf("envp[%d] = %s\n", i, envp[i]);

    return 0;
}

```

```

[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ nano environment_list.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ gcc environment_list.c -o environment_list
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./environment_list
envp[0] = TERM=xterm
envp[1] = SHELL=/bin/bash2
envp[2] = XDG_SESSION_COOKIE=87a47fc7daa86dcdcca015d45654ee83-1601143529.450728-749484223
envp[3] = SSH_CLIENT=88.121.6.235 57654 22
envp[4] = SSH_TTY=/dev/pts/0
envp[5] = USER=ij04115
envp[6] = 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:sg=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:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=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:
envp[7] = MAIL=/var/mail/ij04115
envp[8] = PATH=/opt/anaconda3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
envp[9] = PWD=/users/licence/ij04115/unix_tp2
envp[10] = LANG=fr_FR.UTF-8
envp[11] = SHLVL=1
envp[12] = HOME=/users/licence/ij04115
envp[13] = LS_OPTIONS=--color=auto
envp[14] = LOGNAME=ij04115
envp[15] = SSH_CONNECTION=88.121.6.235 57654 193.49.116.8 22
envp[16] = _=./environment_list
envp[17] = OLDPWD=/users/licence/ij04115
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$

```

```

#include <unistd.h> //execvp
#include <stdio.h> //fprintf, perror
#include <stdlib.h> //exit

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

    int i;

    if(argc == 1) {
        fprintf(stderr, "Utilisation: %s command_patch\n", argv[0]);
        exit(1);
    } //if

    /* echo all environment args - befaure axecvp */
    for( i = 0 ; envp[i] != NULL ; i++ )
        printf("envp[%d] = %s\n", i, envp[i]);

    /* int execvp (const char *file, char *const argv[]); */
    execvp(argv[1], &argv[1]);
    perror(argv[1]); // only get here on error
    exit(2);

} //main

```

```

[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ nano question1_C1.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ gcc question1_C1.c -o question1_C1
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./question1_C1 ./environment_list

```

```

#include <unistd.h> //execve
#include <stdio.h> //fprintf, perror
#include <stdlib.h> //exit

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

    int i;
    char *const new_envp[] = {"PATCH=", NULL}; //patch vide

    if(argc == 1) {
        fprintf(stderr, "Utilisation: %s command_patch\n", argv[0]);
        exit(1);
    } //if

    /* echo all environment args - befaure axecvp */
    for( i = 0 ; envp[i] != NULL ; i++ )
        printf("envp[%d] = %s\n", i, envp[i]);

    /* int execve (const char *patch, char *const argv[], char *const anvp[]); */
    execve(argv[1], &argv[1], new_envp);
    perror(argv[1]); // only get here on error
    exit(2);

} //main

```

```

[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ nano question1_C2.c
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ gcc question1_C2.c -o question1_C2
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./question1_C2
Utilisation: ./question1_C2 command_patch
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$ ./question1_C2 ./environment_list
envp[0] = TERM=xterm
envp[1] = SHELL=/bin/bash2
envp[2] = XDG_SESSION_COOKIE=87a47fc7daa86dcdcca015d45654ee83-1601143529.450728-749484223
envp[3] = SSH_CLIENT=88.121.6.235 57654 22
envp[4] = SSH_TTY=/dev/pts/0
envp[5] = USER=ij04115
envp[6] = 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
=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc
:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7
*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:
01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.
1;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35:*.
1;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:
=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;3
p4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;
lc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35
=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36
3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;3
envp[7] = MAIL=/var/mail/ij04115
envp[8] = PATH=/opt/anaconda3/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:
es
envp[9] = PWD=/users/licence/ij04115/unix_tp2
envp[10] = LANG=fr_FR.UTF-8
envp[11] = SHLVL=1
envp[12] = HOME=/users/licence/ij04115
envp[13] = LS_OPTIONS=--color=auto
envp[14] = LOGNAME=ij04115
envp[15] = SSH_CONNECTION=88.121.6.235 57654 193.49.116.8 22
envp[16] = _=./question1_C2
envp[17] = OLDPWD=/users/licence/ij04115
envp[0] = PATCH=
[ij04115@saphyr:~/unix_tp2]:sam. sept. 26$

```

**Au cours d'un recouvrement, les caractéristiques suivantes ne sont pas conservées selon les conditions spécifiées :**

- **Propriétaire effectif** : si le **set-uid bit** est positionné sur le fichier exécutable chargé, le propriétaire de ce fichier devient propriétaire effectif du processus,
- **Groupe effectif** : si le **set-gid bit** est positionné sur le fichier exécutable chargé, le groupe propriétaire de ce fichier devient groupe propriétaire effectif du processus,
- **Descripteur de fichier ouvert** : si le bit **FD\_CLOEXEC** d'un descripteur a été positionné à l'aide de la primitive **fcntl()**, ce descripteur est fermé après un recouvrement.

**d Ecrire des programmes qui vérifient les affirmations précédentes.**

### Autres identifiants

```
#include <unistd.h>
```

- `pid_t getppid(void);`  
Renvoie le PID du père du processus appelant: *qui est mon père ?*
- `uid_t getuid(void);`  
Renvoie le groupe réel (RGUID) du processus appelant
- `uid_t geteuid(void);`  
Renvoie le groupe effectif (EGUID) du processus appelant
- `uid_t getgid(void);`  
Renvoie le groupe réel (RGUID) du processus appelant
- `uid_t getegid(void);`  
Renvoie le groupe effectif (EGUID) du processus appelant

### Remarque

Aucune de ces primitives ne retourne d'erreur

```
#include <unistd.h> //getuid, geteuid, getgid, getegid
#include <stdio.h> //printf

int main(int argc, char *argv[]) {

    /* Aucune de ces primitives ne retourne d'erreur */

    printf("UID reel : %d\n", getuid() );
    printf("UID effectif : %d\n", geteuid() );

    printf("\n");

    printf("GID reel : %d\n", getgid() );
    printf("GID effectif : %d\n", getegid() );

    return (0);

}
```

```
[ij04115@saphyr:~/unix_tp2]:dim. sept. 27$ nano test_questionC.c
[ij04115@saphyr:~/unix_tp2]:dim. sept. 27$ gcc test_questionC.c -o test_questionC
[ij04115@saphyr:~/unix_tp2]:dim. sept. 27$ ./test_questionC
UID reel : 11978
UID effectif : 11978

GID reel : 3015
GID effectif : 3015
```

**NAME**

fcntl.h - file control options

**SYNOPSIS**

```
#include <fcntl.h>
```

**DESCRIPTION**

The `<fcntl.h>` header shall define the following symbolic constant used for the `fcntl()` file descriptor flags, which shall be suitable for use in `#if` preprocessing directives.

**FD\_CLOEXEC**

Close the file descriptor upon execution of an exec family function.

**NAME**

fcntl - file control

**SYNOPSIS**

```
#include <fcntl.h>
```

```
int fcntl(int fd, int cmd, ...);
```