L2 Informatique

Systèmes d'exploitation I

Exécution de programme

Jessica BECHET

Arguments en ligne de commande

Arguments en ligne de commande

- `main` peut prendre des arguments en ligne de commande

- Invoquer l'exécutable d'un programme avec des arguments

```
$./mon_programme argument1 argument2
```

Arguments en ligne de commande

- Récupérer les arguments dans le programme C :
  - `argc` : entier qui donne le nombre d'arguments+1
  - `argv` : tableau de chaînes de caractères contenant
    - `argv[0]` : nom d'appel du programme
    - `argv[1], argv[2], ...` : chaînes de caractères contenant les arguments passés en ligne de commande
    - `argv[argc]` : `NULL`

```c
#include <stdio.h>

int
main (int argc, char *argv[])
{
 /* Instructions */
}
```

Arguments en ligne de commande

- On préfèrera mettre :
  - `return EXIT_SUCCESS`
    - `main` s'est terminé correctement
    - Par défaut `EXIT_SUCCESS = 0`
  - `return EXIT_FAILURE`
    - Il y a eu un problème dans le `main`
    - Par défaut `EXIT_FAILURE = 1`

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

int
main (int argc, char *argv[])
{
      /* Instructions */
      return EXIT_SUCCESS;
}
```

- Constantes définies dans `<stdlib.h>`

- Portabilité

Primitives algorithmiques

- Arguments en ligne de commande
  - Tableau des arguments : donnée d'entrée du programme principal
  - Même logique qu'en C, mais on ne se préoccupe pas du type

Arguments en ligne de commande

- Exemple 1 : test d'affichage des arguments donnés à un **`main`**

```
Programme principal
Entrée : args
Début
        nargs = len(args)
        Si nargs = 1 alors
                Ecrire "Le programme n'a reçu aucun argument"
        Sinon si nargs >= 2 alors
                Ecrire "Le nom de l'exécutable est :", args[0]
                Ecrire "Le programme a reçu les arguments suivants :"
                Pour i = 1 à nargs-1 faire
                        Ecrire "-->Argument",i, "= ", args[i]
                Finpour
        Finsi
Fin
```

Arguments en ligne de commande

- Exemple 1 : test d'affichage des arguments donnés à un **`main`**

```
int
main(int argc, char *argv[]) {
        if (argc == 1)
                printf("Le programme n'a reçu aucun argument");
        else if (argc >= 2) {
                printf("Le nom de l'exécutable est : %s\n", argv[0]);
                printf("Le programme a reçu les arguments suivants : \n");
                for (int i=1; i < argc; i++)
                        printf("-->Argument %d = %s\n", i, argv[i]);
        }

        return EXIT_SUCCESS;
}
```

```
etudiant@LinuxVM:~/ProgSys/Exemples$ ./exemple_main_arguments
Le programme n'a reçu aucun argument
etudiant@LinuxVM:~/ProgSys/Exemples$ ./exemple_main_arguments a b c
Le nom de l'exécutable est : ./exemple_main_arguments
Le programme a reçu les arguments suivants :
-->Argument 1 = a
-->Argument 2 = b
-->Argument 3 = c
```

Arguments en ligne de commande

- Exemple 2
  - Test avec affichage des arguments donnés à un **`main`**
  - Ajout d'une terminaison **`EXIT_FAILURE`** dans le cas où aucun argument n'a été donné
  - Dans le shell, pour vérifier comment le programme s'est terminé, on peut taper : `echo $?` juste après l'exécution

Arguments en ligne de commande

- Exemple 2

```
Programme principal
Entrée : args
Début
	nargs = len(args)
	Si nargs = 1 alors
		Ecrire "Le programme n’a reçu aucun argument"
		Renvoyer END_NOK
	Sinon si nargs >= 2 alors
		Ecrire "Le nom de l’exécutable est :", args[0]
		Ecrire "Le programme a reçu les arguments suivants :"
		Pour i = 1 à nargs-1 faire
			Ecrire "-->Argument",i, "= ", args[i]
		Finpour
	Finsi
	Renvoyer END_OK
Fin
```

Arguments en ligne de commande

- Exemple 2

```
int
main(int argc, char *argv[]) {
        if (argc == 1) {
                printf("Le programme n'a reçu aucun argument");
                return EXIT_FAILURE;
        } else if (argc >= 2) {
                printf("Le nom de l'exécutable est : %s\n", argv[0]);
                printf("Le programme a reçu les arguments suivants : \n");
                for (int i=1; i < argc; i++)
                        printf("-->Argument %d = %s\n", i, argv[i]);
        }

        return EXIT_SUCCESS;
}
```

Exécution incorrecte
(`EXIT_FAILURE = 1`)

```
etudiant@LinuxVM:~/ProgSys/Exemples$ ./exemple_main_arguments2
Le programme n'a reçu aucun argument
etudiant@LinuxVM:~/ProgSys/Exemples$ echo $?
1
etudiant@LinuxVM:~/ProgSys/Exemples$ ./exemple_main_arguments2 a b c
Le nom de l'exécutable est : ./exemple_main_arguments2
Le programme a reçu les arguments suivants :
-->Argument 1 = a
-->Argument 2 = b
-->Argument 3 = c
etudiant@LinuxVM:~/ProgSys/Exemples$ echo $?
0
```

Exécution correcte
(`EXIT_SUCCESS = 0`)

Options

Options simples – SUSv4

- Précédées d'un tiret (`-`)
- Représentées par un caractère (alphanumérique) simple
- Possibilité de regrouper plusieurs options : `-x -y -z` ≡ `-xyz`
- `--` : (2 tirets) option spéciale pour indiquer la fin de la liste des options
- Tiret isolé = *argument*, pas option
- Bonne pratique lors de l'appel : d'abord la liste des options avant celle des arguments (mais GlibC réordonne au besoin)

Options simples – SUSv4

- Invoquer l'exécutable d'un programme avec des arguments et des options

```
$./mon_programme options argument1 argument2
```

Options simples – SUSv4

```
#include <unistd.h>

int getopt(int argc, char * const argv[], const char *optstring);
extern char *optarg;
extern int optind,opterr, optopt;
```

Options simples – SUSv4

- **`getopt(3)`** : récupérer les options
  - Paramètres : **`argc`** et **`argv`** tels que reçus par le **`main`**, **`optsring`** : chaîne de caractères des options valides (si une option prend un argument on la fait suivre de « **`:`** »)
  - Renvoie le caractère de l'option en cours
  - Renvie « **`?`** » en cas d'option invalide
- **`optind`** : rang du premier élément de **`argv`** qui n'est pas une option
- **`optopt`** : si un caractère non contenu dans **`optstring`** est fourni, un message d'erreur s'affiche et le caractère est inscrit dans **`optopt`** et **`getopt(3)`** renvoie « **`?`** »
- **`opterr`** : si fixé à **`0`**, pas de message d'erreur affiché (pour le cas mentionné plus haut)
- **`optarg`** : accéder à l'argument de certaines options (pointeur sur un élément de **`argv`**)

Options simples – SUSv4

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

#define OPTIONS "ab:c"

int
main(int argc, char *argv[]) {
    opterr = 0;
    char option = 0;
    while(option != -1) {
        option = getopt(argc, argv, OPTIONS);
        switch(option) {
            case 'a': printf("Option a\n");
                      break;
            case 'b': printf("Option b, argument : %s\n", optarg);
                      break;
```

Options simples – SUSv4

```
                case 'c': printf("Option c\n");
                          break;
                case '?': printf("Option %c inconnue\n", optopt);
                          break;
            }
        }
        printf("Indice du premier argument non option : %d\n", optind);
        printf("Liste des arguments non option : ");
        for (int i = optind; i < argc; i++)
            printf("%s ", argv[i]);
        printf("\n");
        return EXIT_SUCCESS;
}
```

Options simples – SUSv4

Attention : l'argument attendu pour l'option **b** est la chaîne de caractères qui suit directement l'option.

```
jessica@jessica-VM:~$ ./main -a -b -c -x xyz tuv wxy
Option a
Option b, argument : -c
Option x inconnue
Indice du premier argument non option : 5
Liste des arguments non option : xyz tuv wxy
jessica@jessica-VM:~$ ./main -a -b hub -c -x xyz tuv wxy
Option a
Option b, argument : hub
Option c
Option x inconnue
Indice du premier argument non option : 6
Liste des arguments non option : xyz tuv wxy
jessica@jessica-VM:~$ ./main -abcx hub xyz tuv wxy
Option a
Option b, argument : cx
Indice du premier argument non option : 2
Liste des arguments non option : hub xyz tuv wxy
jessica@jessica-VM:~$ ./main -acxb hub xyz tuv wxy
Option a
Option c
Option x inconnue
Option b, argument : hub
Indice du premier argument non option : 3
Liste des arguments non option : xyz tuv wxy
```

Options longues – Gnu

- Précédées de deux tirets (`--`)
- Représentées par un nom complet
- `get_optlong(3)`

Exécution d’un programme

Primitives algorithmiques

- Exécuter un programme

```
ret = exec(execfile, parameters, [environment])
```

```
/*
 * Lancement de la commande exec pour le repertoire parent
 */
Programme principal
Début
        /* Execution ls */
        params = {"/bin/ls", ".."}
        exec(ls, params)
        /* On ne revient qu'en cas d'erreur de la fonction exec */
        Ecrire "Erreur exec"

Fin
```

Exécution d'un programme

- Lancer une application : famille **`exec(3)`**
  - **Remplace** l'espace mémoire du processus appelant par le code et les données de la nouvelle application
    - Retour en cas d'erreur : **`-1`**
    - Sinon processus appelant entièrement remplacé
    - Réussite : on ne revient pas dans le programme

Exécution d'un programme

- Lancer une application : famille **exec(3)**
  - L'ancien programme **transmet** automatiquement au nouveau :
    - PID, PPID, etc.
    - UID, GID (sauf utilisation set-UID, set-GID)
    - Signaux : masques des signaux bloqués, signaux en attente, signaux ignorés
    - Descripteurs de fichiers ouverts + verrous (sauf utilisation « close-on-exec »)
    - Temps d'exécution non remis à 0
    - Privilèges identiques (sauf utilisation set-UID)

Exécution d'un programme

- Lancer une application : famille **`exec(3)`**
  - Six variantes : **`execl(3), execle(3), execlp(3), execv(3), execve(2) et execvp(3)`**
    - Suffixe commence par "**`l`**" : utilise une liste d'arguments à transmettre (de nombre variable)
    - Suffixe commence par "**`v`**" : utilise un tableau à la manière de **`argv[]`**
    - Se termine par "**`e`**" : environnement explicitement passé dans les arguments de la fonction dans un tableau **`envp[]`** ; les autres utilisent la variable **`environ`**
    - Se termine par "**`p`**" : utilise la variable d'environnement **`PATH`** pour rechercher le répertoire dans lequel se situe l'application à lancer, les autres nécessitent un chemin d'accès complet
      - Lister les chemins en les séparant par des « **`:`** »
      - Toujours commencer par les répertoires contenant les applications les plus utilisées

Exécution d'un programme

- Lancer une application
  - Seul **`execve(2)`** est un appel-système
  - Les autres sont implémentés à partir de ce-dernier

Exécution d'un programme

- Lancer une application : **execve(2)**

```
#include <unistd.h>

int execve (const char *file, char *const params[], char *const envp[]);
```

- Lancer une application : famille **execv(3)**

```
#include <unistd.h>

int execv (const char *path, char *const params[]);
int execvp (const char *file, char *const params[]);
```

Exécution d'un programme

- `path` : chemin complet vers l'application à lancer
- `file` : nom de l'application à lancer
- `envp[]` : environnement sous forme de tableau
  - Se termine par **`NULL`**
  - Lorsqu'il n'est pas requis en argument de la fonction, variable **`environ`** externe globale utilisée
- Famille **`execv()`** : liste des arguments dans le vecteur **`params[]`**
  - Contient des chaînes de caractères correspondant aux arguments
  - **`params[0]`** = nom de l'application à lancer (sans chemin)
  - Se termine par **`NULL`**

Exécution d'un programme

- Exemple : **`execve(2)`**
  - Exécution de la commande **`ls`** sur le répertoire parent (de celui courant)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

extern char ** environ;

int
main(int argc, char *argv[]) {
        char *params[] = {"ls", "..", NULL};

        execve("/bin/ls", params, environ);

        /* Si execve fonctionne, on ne revient pas ici */
        perror("execve");

        return EXIT_FAILURE;
}
```

Exécution d'un programme

- Exemple : **`execv(3)`**
  - Exécution de la commande **`ls`** sur le répertoire parent (de celui courant)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int
main(int argc, char *argv[]) {
        char *params[] = {"ls", "..", NULL};

        execv("/bin/ls", params);

        /* Si execv fonctionne, on ne revient pas ici */
        perror("execv");

        return EXIT_FAILURE;
}
```

Exécution d'un programme

- Exemple : **`execvp(3)`**
  - Exécution de la commande **`ls`** sur le répertoire parent (de celui courant)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int
main(int argc, char *argv[]) {
        char *params[] = {"ls", "..", NULL};

        execvp("ls", params);

        /* Si execvp fonctionne, on ne revient pas ici */
        perror("execvp");

        return EXIT_FAILURE;
}
```

Exécution d'un programme

- Lancer une application : famille **`execl(3)`**

```
#include <unistd.h>

int execl (const char *path, const char *arg, ..., NULL);
int execlp (const char *file, const char *arg, ..., NULL);
int execle (const char *path, const char *arg, ..., NULL, const char *envp[]);
```

Exécution d'un programme

- `path` : chemin complet vers l'application à lancer
- Famille `execl(3)` : liste des arguments séparés par une virgule et terminés par `NULL`
- `file` : nom de l'application à lancer
- `envp[]` : environnement sous forme de tableau

Exécution d'un programme

- Exemple : **execl(3)**
  - Exécution de la commande **ls** sur le répertoire parent (de celui courant)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int
main(int argc, char *argv[]) {
        execl("/bin/ls", "ls", "..", NULL);

        /* Si execl fonctionne, on ne revient pas ici */
        perror("execl");

        return EXIT_FAILURE;
}
```

Exécution d'un programme

- Exemple : **`execle(2)`**
  - Exécution de la commande **`ls`** sur le répertoire parent (de celui courant)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

extern char ** environ;

int
main(int argc, char *argv[]) {
        execle("/bin/ls", "ls", "..", NULL, environ);

        /* Si execle fonctionne, on ne revient pas ici */
        perror("execle");

        return EXIT_FAILURE;
}
```

Exécution d'un programme

- Exemple : **`execlp(3)`**
  - Exécution de la commande **`ls`** sur le répertoire parent (de celui courant)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

int
main(int argc, char *argv[]) {
        execlp("ls", "ls", "..", NULL);

        /* Si execlp fonctionne, on ne revient pas ici */
        perror("execlp");

        return EXIT_FAILURE;
}
```

Exécution d’un programme

Un petit mot sur l’environnement

Un petit mot sur l'environnement

- Variables d'environnement
  - Chaînes de caractères avec affectations : `NOM=VALEUR`
    - Avec différentiation majuscule/minuscule
  - Accessibles par les processus
    - Possibilité pour un processus de modifier/supprimer des variables de son environnement
    - Actions sur le processus en cours et ses descendants mais pas sur son parent
    - `getenv(3)` : rechercher une variable d'environnement
    - `putenv(3)` : créer une variable d'environnement
    - `setenv(3)` : modifier une variable d'environnement
    - `unsetenv(3)` : supprimer une variable d'environnement

  - Possibilité de passer un troisième argument à la fonction main correspondant au tableau d'environnement : mais à éviter pour des raisons de portabilité

Merci !