La commande Kill– Système d’exploitation2 TC

Info (IAP) S4

2024-2025

Pr. Er-rajy Latifa

La commande Kill : définition

La commande kill est utilisée dans les systèmes Unix/Linux pour envoyer des signaux aux processus en cours d'exécution, généralement pour les arrêter.

Syntaxe de base :

kill [options] PID

- **PID** : Identifiant unique du processus cible.
- **options** : Spécifient le type de signal ou d'action.

La commande Kill : Signal

Un signal est un mécanisme de communication entre le système d'exploitation et les processus. Il permet d'envoyer des notifications aux processus pour leur indiquer un événement, comme leur arrêt, leur mise en pause ou leur redémarrage.

- Il est envoyé par le noyau, un utilisateur ou un autre processus.
- Chaque signal est représenté par **un numéro** et **un nom**.

La commande Kill : Signaux

Exemples des Signaux : **kill -l**

Numéro	Nom	Description
1	SIGHUP	Ferme ou redémarre un processus.
2	SIGINT	Interrompt un processus (équivalent à Ctrl + C).
9	SIGKILL	Tue immédiatement un processus.
15	SIGTERM	Demande à un processus de se terminer proprement.
19	SIGSTOP	Met un processus en pause.
18	SIGCONT	Relance un processus mis en pause.
17	SIGCHLD	Envoyé à un processus parent lorsqu'un de ses processus enfants se termine.

La commande Kill : Considérations supplémentaires

1. Prudence avec kill -9 (SIGKILL)

Le signal SIGKILL (-9) force l'arrêt immédiat d'un processus sans lui laisser le temps d'effectuer des **actions de nettoyage** (fermeture des fichiers ouverts, sauvegarde des données, libération de mémoire, etc.).

2. Permissions nécessaires

Par défaut, un utilisateur ne peut pas tuer un processus appartenant à un autre utilisateur, sauf si :

- Il est root (sudo est nécessaire).
- Il a été accordé des permissions spécifiques.

La commande Kill : Syntaxe

1. kill -<signal>, <PID> ou kill --signal <signal> <PID> :

pour Envoyer le signal SIGKILL (9) pour forcer l'arrêt du processus 5678

kill -KILL 5678 ou **kill --signal KILL 5678**

2. kill -<numéro_signal> <PID>:

kill -9 5678

1. Lancez manuellement (interface graphique) un éditeur de texte.
2. Cherchez le PID de l'éditeur avec la commande :

pgrep gedit

La commande Kill : Exemples

1. Ouvrez un terminal et lancez top : **top**

2. Ouvrez une seconde console (Ctrl+Alt+T) et trouvez son PID :

pgrep top

3. envoyer le signal de terminaison au processus de top:

kill -TERM 1234

4. Si top ne s'arrête pas, forcez l'arrêt :

kill -KILL 1234

La commande Kill : Arrêter et relancer un processus

1. **Ouvrez un terminal et lancez ping :** **ping google.com**

2. **Ouvrez une seconde console (Ctrl+Alt+T) et trouvez son PID :**

pgrep ping

- **Suspendre un processus** :
  Arrête temporairement l'exécution le processus avec :

kill -STOP PID

- **Reprendre un processus** :
  Relance l'exécution le processus suspendu avec :

kill -CONT PID

La commande Kill : Utilisation de killall

Lancez manuellement firefox :

- **Terminer des processus par leur nom :**

La commande killall arrête tous les processus portant le même nom :

killall firefox

Fonctions d'exécution : Argument

Un argument est une valeur transmise à un programme lors de son exécution en ligne de commande.

Exemple :

./mon_programme arg 1 option1

- **./mon_programme** est le nom de l'exécutable du programme.
- **./mon_programme,** arg1 et option1 sont des arguments.

Fonctions d'exécution : Argument

Syntaxe en C :

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

argc (Argument Count) :

- Représente le **nombre d'arguments** passés au programme, y compris le nom du programme lui-même.

argv[] (Argument Vector) :

- Un tableau de chaînes de caractères contenant les arguments passés au programme.
- **argv[0]** contient toujours le chemin ou le nom du programme exécuté.
- Les autres indices (**argv[1], argv[2], ...**) contiennent les arguments fournis par l'utilisateur.

Fonctions d'exécution : Exercice 1

Ecrire un programme en C qui affiche :

1. Le nombre total d'arguments passés sur la ligne de commande (argc).
2. La valeur de chaque argument dans le tableau argv[].

Fonctions d'exécution : Exercice 1

```c
#include <stdlib.h>  // Inclusion de la bibliothèque standard pour EXIT_SUCCESS
#include <stdio.h>   // Inclusion de la bibliothèque standard pour printf

int main(int argc, char *argv[]) {
    printf("argc = %d\n", argc);  // Affiche le nombre total d'arguments

    for (int i = 0; i < argc; i++) {  // Parcourt tous les arguments
        printf("argv[%d] = %s\n", i, argv[i]);  // Affiche chaque argument avec son index
    }

    return EXIT_SUCCESS;  // Indique que le programme s'est exécuté avec succès
}
```

Sans argument :

gedit test_program.c

gcc -o test_program test_program.c && ./test_program

Fonctions d'exécution : Exercice 1

```c
#include <stdlib.h>  // Inclusion de la bibliothèque standard pour EXIT_SUCCESS
#include <stdio.h>   // Inclusion de la bibliothèque standard pour printf

int main(int argc, char *argv[]) {
    printf("argc = %d\n", argc);  // Affiche le nombre total d'arguments

    for (int i = 0; i < argc; i++) {  // Parcourt tous les arguments
        printf("argv[%d] = %s\n", i, argv[i]);  // Affiche chaque argument avec son index
    }

    return EXIT_SUCCESS;  // Indique que le programme s'est exécuté avec succès
}
```

Avec arguments:
./test_program arg1 arg2 arg3

Fonctions d'exécution : execle()

Exécute un programme en permettant de passer explicitement les variables d'environnement.

Syntaxe :

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

Paramètres :

- **path** : Chemin absolu du programme à exécuter.
- **arg** : la liste des arguments à passer au programme, terminée par NULL
- **envp** : Tableau de pointeurs vers des variables d'environnement, terminé par NULL.

- **char *const argv[]** → Les adresses stockées dans argv sont constantes, mais les chaînes peuvent être modifiées.
- **const char *argv[]** → Les chaînes sont constantes (on ne peut pas modifier argv[i][j]), mais les adresses dans argv peuvent être modifiées.

Fonctions d'exécution : Exercice 2

Écrivez un programme en langage C qui exécute la commande **ls -l** en utilisant la fonction **execle()**.
Votre programme doit :

1. Définir un environnement personnalisé contenant au moins les variables **PATH et HOME**.
2. Utiliser execle() pour exécuter /bin/ls avec l'argument -l.

Fonctions d'exécution : Exercice 2

```c
#include <unistd.h>

int main() {
    char *env_vars[] = {"PATH=/usr/bin", "HOME=/home/user", NULL};
    execle("/bin/ls", "ls", "-l", NULL, env_vars);
    return 0;
}
```

```
gedit test_execle.c
gcc -o test_ execle test_ execle.c && ./test_ execle
```

Fonctions d'exécution : execlp()

Recherche automatiquement le chemin du programme dans la variable d'environnement **PATH.**

Syntaxe :

```
int execlp(const char *file, const char *arg, ..., (char *) NULL);
```

Paramètres :

- **file** : le nom du fichier exécutable (peut être un nom de commande, recherché dans le PATH).
- **arg** : la liste des arguments à passer au programme, terminée par NULL.
- ***(char) NULL** : la liste des arguments doit être terminée par NULL.

Fonctions d'exécution : Exercice 3

Écrivez un programme en langage C qui utilise la fonction **execlp** pour exécuter la commande **ls -l**.

```c
#include <unistd.h>

int main() {
    execlp("ls", "ls", "-l", NULL);
    return 0;
}
```

```
gedit test_execlp.c
gcc -o test_ execlp  test_ execlp.c && ./test_ execlp
```

Fonctions d'exécution : execv()

Passe les arguments du programme via un **vecteur de pointeurs** (tableau de chaînes).

Syntaxe :

```
int execv(const char *path, char *const args[]);
```

Paramètres :

- **path** : Chemin absolu du programme à exécuter.
- **args** : la liste des arguments à passer au programme, terminée par NULL

Fonctions d'exécution : Exercice 3

Écrivez un programme en langage C qui exécute la commande **ls -l** en utilisant la fonction **execv()**.

```c
#include <unistd.h>

int main() {
    char *args[] = {"ls", "-l", NULL};
    execv("/bin/ls", args);
    return 0;
    }
```

```
gedit test_execv.c
gcc -o test_ execv  test_ execv.c && ./test_ execv
```

Fonctions d'exécution : execvp()

Similaire à execv(), mais recherche automatiquement le chemin dans PATH.

Syntaxe :

```
int execvp(const char *file, char *const args[]);
```

Paramètres :

- **file** : le nom du fichier exécutable.
- **args** : Un tableau de chaînes de caractères représentant la liste des arguments passés au programme. Il doit être terminée par NULL.

Fonctions d'exécution : Exercice 4

Écrivez un programme en langage C qui exécute la commande **ls -l** en utilisant la fonction **execvp()**.

```c
#include <unistd.h>

int main() {
    char *args[] = {"ls", "-l", NULL};
    execvp("ls", args);
    return 0;
}
```

```
gedit test_execvp.c
gcc -o test_ execvp  test_ execvp.c && ./test_ execvp
```