INF2132 : Systèmes d'Exploitation

TP5 - APIs POSIX et Appels Système sous Linux

Nom de l'enseignant :
Pr. Ilias Tougui

Nom de l'assistant :
Pr. Yasser Aderghal

Table des matières

1 Objectifs Pédagogiques **2**

2 Le Manuel Linux : man **2**

3 Pré-requis et matériel **2**

4 Préparation de l'Environnement Linux **3**

5 Analyse du Code Source myls.c **4**

6 Compilation et Traçage des Appels Système **5**

Lisez attentivement cette page

1 Objectifs Pédagogiques

À la fin de cet TP, vous serez capable de :

* Compiler un programme C utilisant les APIs POSIX de gestion des répertoires
* Comprendre la relation entre les APIs POSIX et les appels système Linux correspondants
* Analyser les appels système avec l'outil `strace`
* Consulter la documentation système avec les pages `man`
* Installer une commande personnalisée dans le système

2 Le Manuel Linux : man

Le manuel Linux est organisé en sections numérotées. Pour consulter les pages du manuel, utilisez la commande :

```
$ man [section] commande
```

Sections principales :

* Section 1 : Commandes utilisateur
* Section 2 : Appels système (fournis par le noyau)
* Section 3 : Fonctions de bibliothèque (API)
* Section 5 : Formats de fichiers
* Section 8 : Commandes d'administration

3 Pré-requis et matériel

* Système Linux/Unix
* Fichier fourni : `myls.c`
* Comprehension du cours : **CM4 : Services et Interfaces d'un Système d'exploitation**

Pour les étudiants qui n'ont pas installé WSL ni de machine virtuelle Ubuntu, il est possible d'utiliser temporairement :

* `https://cocalc.com/features/terminal`
* **Termux** sur Android
* **iSH Shell** sur IOS

4 Préparation de l'Environnement Linux

Durée : 20 min, Note : 5 points

En utilisant les commandes : `pwd`, `ls`, `cd`, `mkdir`, `touch`, `nano` et `cat`, suivez les consignes ci dessous pour créer l'environnement de travail structuré de ce TP.

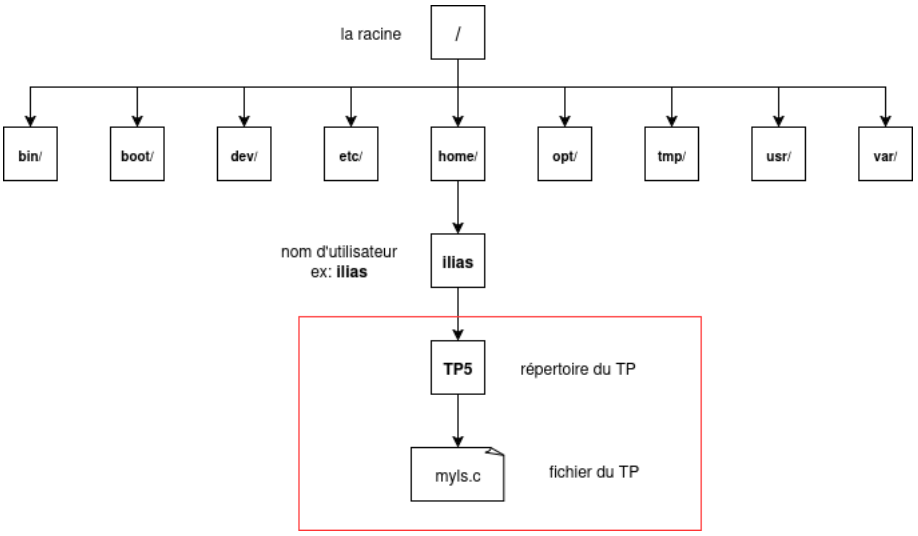


FIGURE 1 – Structure de l'environement de travail.

Exercice

1. Déplacez-vous dans le répertoire racine (/) à l'aide de la commande `cd`.
2. Ensuite, Déplacez-vous dans votre répertoire personnel (~).
3. Créez un nouveau répertoire nommé `TP5` en utilisant la commande `mkdir`.
4. Accédez au répertoire `TP5`.
5. Créez un fichier de code source (`myls.c`) à l'aide de la commande `touch`.
6. Vérifiez la création du fichier en utilisant la commande `ls`.
7. Remplissez le fichier avec le contenu indiqué sur la plateforme `Connect` en utilisant l'éditeur `nano`.
8. Enregistrez vos modifications dans `nano` à l'aide des touches `Ctrl + O`, puis `Entrée`, et fermez l'éditeur avec `Ctrl + X`.
9. Affichez le contenu du fichier dans le terminal avec la commande `cat`.
10. Lisez attentivement le contenu de `myls.c`.
11. Appelez le professeur pour lui montrer le travail acompli.

5 Analyse du Code Source myls.c

Durée : 20 min, Note : 5 points

1. Qu'est-ce que le terminal ? Quelle est sa relation avec le shell ?
2. Pourquoi les commandes `pwd`, `ls`, `cd`, `mkdir`, `touch`, `nano` et `cat` fonctionnent-elles depuis le terminal ?
3. Quelle est la différence entre une commande intégrée (*built-in*) et une commande externe ? Donnez un exemple de chaque type.
4. Comment pouvez-vous vérifier si une commande est intégrée ou externe ? (Indice : utilisez `type nom_commande`)
5. Assurez-vous d'être dans le répertoire `TP5`.
6. Comptez le nombre de lignes du fichier `myls.c` avec la commande `wc -l`.
7. Affichez les 30 premières lignes du fichier `myls.c`
8. Identifiez tous les fichiers d'en-tête inclus au début du code (lignes commençant par `#include`).
9. Dans le code source, recherchez en utilisant la commande `grep` les fonctions API POSIX suivantes :
   * `opendir()`
   * `readdir()`
   * `malloc()`
   * `closedir()`
10. Consultez la documentation des APIs POSIX identifiées :

```
$ man 3 opendir
$ man 3 readdir
$ man 3 malloc
$ man 3 closedir
```

11. Que fait la fonction `format_size()` ? Quels paramètres prend-elle ?
12. Que fait la fonction `show_help()` ? Pourquoi est-elle utile ?
13. Appelez le professeur pour vérifier votre compréhension

Point Clé à Retenir

Le fichier source `.c` contient des appels aux **fonctions de l'API POSIX** (comme `opendir()`, `readdir()`, `stat()`). Ces fonctions sont définies dans les fichiers d'en-tête (`<dirent.h>`, `<sys/stat.h>`). Lors de la compilation, le programme sera lié avec la bibliothèque C (libc) qui contient l'implémentation de ces APIs. À l'exécution, ces fonctions API invoqueront les **appels système réels** du noyau Linux.

6 Compilation et Traçage des Appels Système

Durée : 20 min, Note : 5 points

Les fonctions API POSIX que vous avez identifiées dans l'exercice précédent (`opendir()`, `readdir()`, `stat()`) sont implémentées dans la bibliothèque C (libc). Lors de l'exécution, ces fonctions invoqueront les appels système réels du noyau Linux.

1. Assurez-vous d'être dans le répertoire `TP5`.
2. Compilez le programme `myls.c` avec la commande suivante :

```
$ gcc -o myls myls.c -Wall -Wextra
```

3. Que font les options `-Wall` et `-Wextra` lors de la compilation ?
4. Vérifiez que la compilation a réussi en listant les fichiers du répertoire.
5. Testez votre commande localement :

```
$ ./myls
```

6. Affichez l'aide de votre commande :

```
$ ./myls --help
```

7. Testez avec différentes options :

```
$ ./myls -l
$ ./myls -lh
$ ./myls --size
```

8. Installez la commande dans le système pour pouvoir l'utiliser partout :

```
$ sudo cp myls /usr/local/bin/
$ sudo chmod +x /usr/local/bin/myls
$ which myls
```

9. Vérifiez l'installation en exécutant depuis votre répertoire personnel :

```
$ cd
$ myls
```

10. Utilisez `strace` pour observer les appels système :

```
$ cd ~/TP5
$ strace -o trace_myls.txt myls
```

11. Affichez le fichier de trace :

```
$ <commande> trace_myls.txt
```

12. Recherchez les appels système correspondant aux APIs POSIX :

```
$ grep "openat\|getdents64\|newfstatat\|brk\|mmap\|close" trace_myls.txt
```

13. Consultez la documentation des appels système identifiées :

```
$ man 2 openat
$ man 2 getdents64
$ man 2 newfstatat
...
```

14. Comptez le nombre d'appels système :

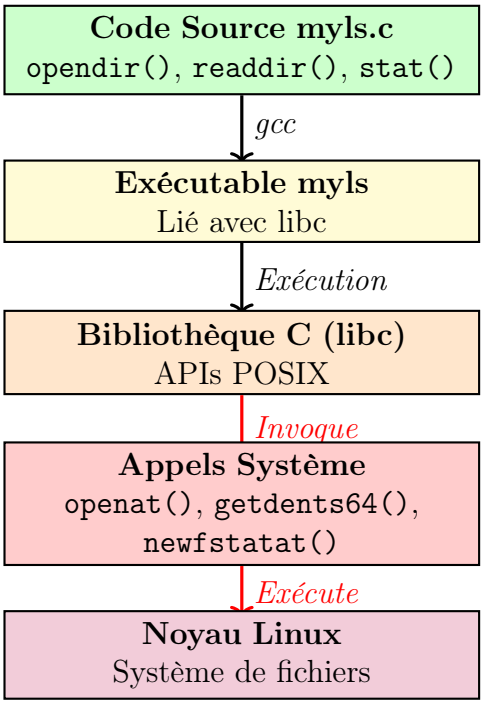
```
$ grep "getdents64" trace_myls.txt | wc -l
...
```

15. Comparez avec la commande système `ls` :

```
$ strace -c myls
$ strace -c ls
```

16. Quelle API POSIX est utilisée pour ouvrir un répertoire dans le code source ?
17. Quel appel système Linux correspond à `opendir()` ? (Observable dans `strace`)

Schéma Récapitulatif



Note : `strace` montre uniquement les appels système (en rouge).