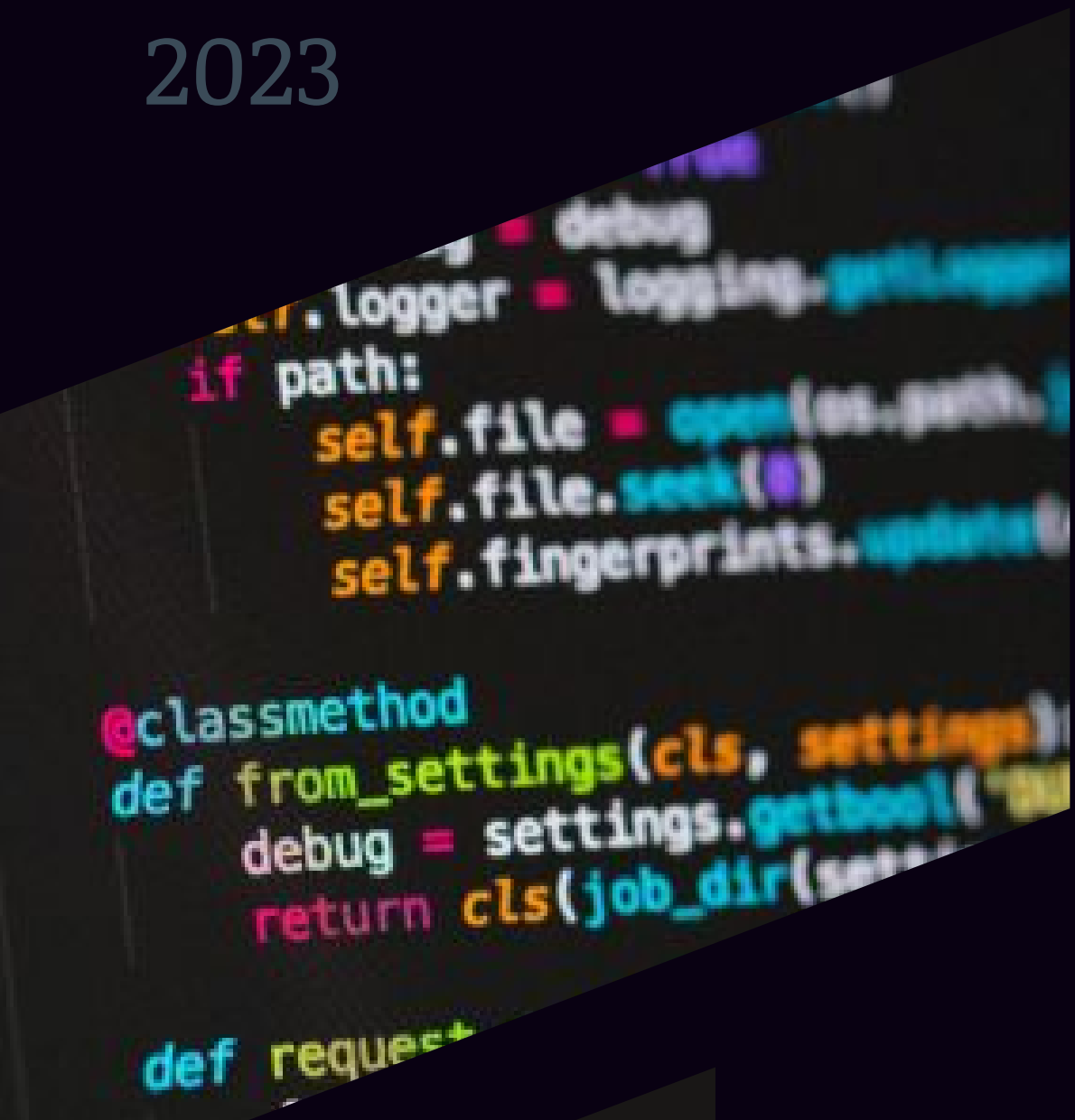
RAPPORT: PROJET OS

TECHNOLOGIE DE L'INFORMATIQUE

2023

Brice Delcroix
Loan Istas
Samin Fawad

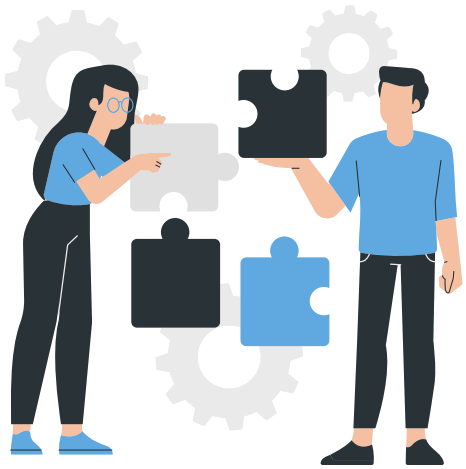
1TM2



Mme Vroman
et
Mme Masson

EPHEC

Sommaire



1.Introduction

Ce rapport présente le développement d'un programme en langage C sous Linux, réalisé par le groupe composé de Loan Istas, Brice Delcroix et Samin Fawad. Ce programme vise à traiter des commandes, accompagnées de leurs options et paramètres, fournies via la ligne de commande. De plus, une évolution majeure du projet consiste à prendre en charge l'exécution de shell scripts, c'est-à-dire des fichiers contenant une suite de commandes shell.

Ce rapport détaillera en profondeur l'évolution du projet ainsi que les techniques utilisées pour sa réalisation. Nous partagerons également nos observations et les contraintes auxquelles nous avons été confrontés. L'avancement de ce travail a été réalisé par le groupe lors de réunions régulières, en combinant nos recherches individuelles afin d'optimiser notre efficacité.

2.Analyse du travail

2.1 Première partie du programme

Notre programme vise à récupérer une commande, à créer un processus fils qui l'exécute et renvoie le résultat au processus père qui se charge de l'afficher.

2.1.1 Bibliothèques : Les outils indispensables!

Nous déclarons maintenant les différentes bibliothèques accompagnées de la constante BUFFER_SIZE qui indique que chaque occurrence de cette constante dans le code est remplacé par 1024.

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

#define BUFFER_SIZE 1024
```

stdio.h et stdlib.h sont les bibliothèques requise pour les programmes c, nous les utilisons depuis le début de l'année. String.h permet de manipuler les chaînes de caractères telles que la concaténation, la comparaison, la copie et la recherche.

La bibliothèque unistd.h fournit un accès à des fonctionnalités système de bas niveau telles que les appels système, les constantes, et les fonctions d'entrée/sortie.

Enfin sys/wait.h est intéressante car elle nous fournit des fonctions pour la gestion des processus enfants, telles que la suspension, l'attente de leur terminaison, la récupération de leurs états et la gestion de leurs signaux.

2.1.2 Variables utiles

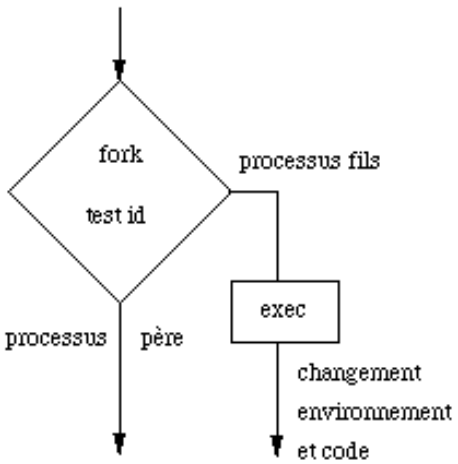
Par la suite, nous récupérons la commande passée en argument via la fonction "main()". Nous déclarons ensuite le tableau de caractères de taille BUFFER_SIZE, "pid", "status" et "pipefd[2]".

Grâce à la condition "argc<2" nous vérifions qu'une commande à bien été passée en argument. On ouvre ensuite le canal de communication entre le processus fils et le processus père. Nous en profitons pour vérifier que le canal a bien été créé

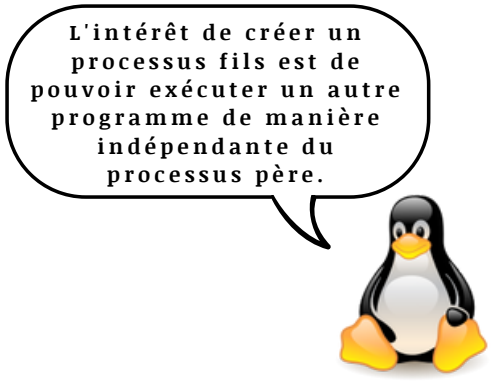
```
ichiky@ichiky-virtual-machine:~$ ./projet
Usage: ./projet <command> [arg1] [arg2] ...
```

2.1.3 Processus en action

Nous créons un processus fils grâce à la fonction "fork()" et récupérons son identifiant dans la variable "pid" qui nous permet d'accéder aux différents processus.



La condition suivante vérifie si nous sommes dans le processus fils, dans le processus père ou s'il s'agit d'une erreur.
Dans le processus fils, Nous redirigeons la sortie vers le canal de communication et on exécute la commande passée en argument.
Dans le processus père, Nous lisons le résultat via l'extrémité de lecture du canal et on attend la fin du processus fils.
Pour finir, Nous affichons les résultats et on quitte le programme.



2.2 Deuxième partie du programme

Nous passons maintenant à une étape plus avancée où une simple commande ne suffit plus. Nous devons maintenant prendre en compte un ou plusieurs fichiers shell qui contiennent des séquences de commandes. Au lieu d'exécuter une seule commande, notre programme devra être en mesure d'exécuter chaque commande présente dans ces fichiers et de traiter leurs résultats. Cela représente un défi supplémentaire, mais nous sommes prêts à relever cette nouvelle étape du projet.

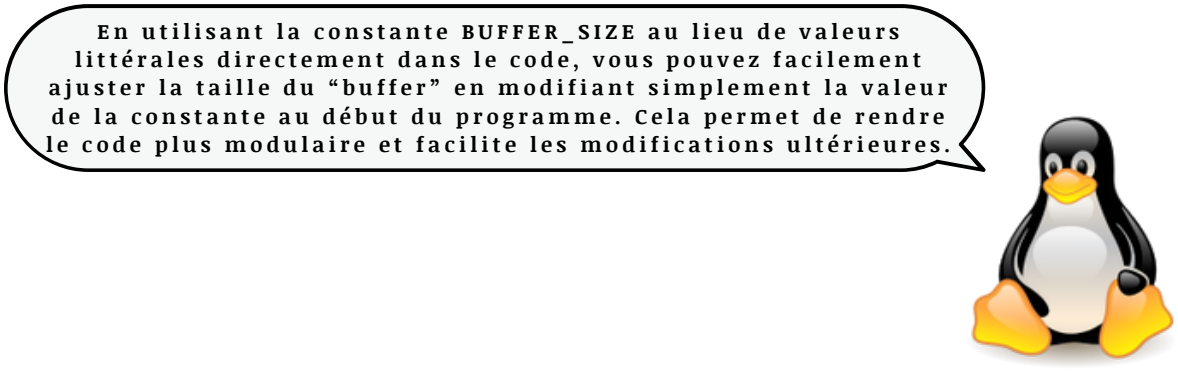
2.2.1 Début du programme : les inchangés

Le début du programme n'a pas beaucoup changé, aucune bibliothèque supplémentaire n'a été ajoutée. La constante BUFFER_SIZE est toujours présente ainsi que les autres variables globales, sauf le tableau buffer[] qui a été déplacé.

Nous avons organisé notre programme en deux boucles :

- Une qui gère chaque fichier shell et l'ouvre, on y défini également le tableau de caractères buffer[] pour qu'il se réinitialise à chaque itération.

- L'autre qui est charger d'exécuter chaque ligne du fichier shell.



2.2.2 Nouveautés de la deuxième partie

Dans cette deuxième partie, nous utilisons les fonctions "fopen()", "fgets()" et "fclose()" afin de gérer l'ouverture, la fermeture et la récupération des données du fichier shell.

On utilise la fonction "strcspn()" dans une partie du code, ce qui nous permet de supprimer le caractère de saut de ligne "\n".

Lors de la lecture, nous utilisons une boucle pour lire le résultat de l'exécution. Comme "read()" renvoie le nombre d'octet lu, nous demandons à la boucle d'attendre que tous les résultats soient lus avant de continuer.

2.2.3 Erreur de lecture

Après nos premiers tests du programme, nous avons constaté des erreurs de lecture. Pour remédier à cela, nous devons modifier la condition "total_bytes > 0" en "total_bytes >= 0" car il est possible que le total des bytes soit égal à zéro. Cela indique simplement que le programme a terminé la lecture et qu'il n'y a plus rien à lire.

```
ichiky@ichiky-virtual-machine:~/rapport$ ./projet shell.sh
Le processus fils a retourné pour la commande "ls -o" : total 28
-rwxrwxr-x 1 ichiky 16584 mei 17 19:32 projet
-rw-rw-r-- 1 ichiky  3597 mei 17 19:31 projet.c
-rwxrwxrwx 1 ichiky    28 mei 11 11:43 shell.sh

Le processus fils a terminé avec le code de retour 0.
Hello World
Erreur lors de la lecture du résultat depuis le canal.
Le processus fils a terminé avec le code de retour 0.
projet  projet.c  shell.sh
Erreur lors de la lecture du résultat depuis le canal.
Le processus fils a terminé avec le code de retour 0.
```

2.3 Demonstration

Pour essayer notre programme, nous utilisons le compilateur GCC pour compiler le programme et nous créons un fichier "shell.sh".


projet


projet.c


shell.sh

Notre fichier shell.sh contient les commandes :

- ls -l
- echo “Hello World”
- ls

Après l'erreur de lecture mentionnée auparavant, le programme fonctionne comme nous le voulions.

```
ichiky@ichiky-virtual-machine:~/rapport$ ./projet shell.sh
Le processus fils a retourné pour la commande "ls -o" : total 28
-rwxrwxr-x 1 ichiky 16584 mei 17 19:41 projet
-rw-rw-r-- 1 ichiky  3598 mei 17 19:41 projet.c
-rwxrwxrwx 1 ichiky    28 mei 11 11:43 shell.sh

Le processus fils a terminé avec le code de retour 0.
Hello World
Le processus fils a retourné pour la commande "echo "Hello World"" :
Le processus fils a terminé avec le code de retour 0.
projet  projet.c  shell.sh
Le processus fils a retourné pour la commande "ls" :
Le processus fils a terminé avec le code de retour 0.
```

Le programme est également adapté si nous lui demandons d’exécuter plusieurs fichiers shell.

3.Planning

Dans cette partie, nous présentons le planning détaillé de notre projet, en expliquant les différentes étapes que nous avons suivies. Nous expliquons également comment nous avons géré notre travail en groupe et comment nous avons réparti les tâches entre les différents membres de notre équipe.

Le **19/04**, après avoir analysé les consignes et le travail à réaliser, nous avons déterminé les différentes tâches et étapes à suivre pour mener à bien le projet. Nous avons commencé la première partie du travail en décomposant celle-ci en trois étapes distinctes :

a) Récupérer la commande depuis la ligne de commande et créer la procédure fils qui l'exécute
b) Rediriger la sortie du processus fils pour empêcher son affichage après l'exécution
c) Faire en sorte que le processus fils renvoie les résultats au processus père qui les affiche à l'écran.

Nous avons créé un document Google afin de partager des informations et le planning des réunions qui serviront pour le rapport sur le même document. Nous avons aussi créé un github pour conserver le code. Chaque membre du groupe a réalisé des recherches individuelles.

Le **26/04**, nous avons mis en commun les recherches effectuées et l'avancement de notre travail. Nous avons également suivi un cours portant sur la commande execvp() et les pointeurs, qui nous ont été utiles pour le projet.

Durant la semaine du **03/05**, nous avons avancé sur la première partie du travail et réussi à récupérer la commande, l'envoyer au fils et rediriger la sortie vers le père pour qu'il l'affiche.

Le **10/05**, nous avons effectué des recherches sur la commande Pipe et dub2, puis finalisé la première partie du travail. Nous avons débuté la deuxième partie et réalisé des recherches sur les fichiers shell. Nous avons fixé une réunion le lendemain pour avancer plus rapidement, afin de terminer la deuxième partie.

Le **11/05**, nous avons tenu une réunion de deux heures et terminé ensemble la deuxième partie du travail. Cependant, nous avons constaté des erreurs de lecture étranges. Malgré cela, le programme s'exécute correctement. Nous avons pour projet de finir le rapport pour la semaine suivante et de régler les derniers problèmes avec le programme.

Nous avons travaillé en groupe de manière intensive, en répartissant certaines recherches entre les membres et en avançant simultanément sur le programme. Le rapport a été complété par chaque membre du groupe avec une correction finale réalisée lors de la dernière réunion, afin d'éviter les incohérences.

Le **17/05**, durant le cours nous avons résolu le problème de lecture. Nous travaillons maintenant pleinement sur le rapport et la défense. Un de nous s’occupe de finir le rapport et de le faire corriger, un autre s’occupe de la relecture et de commencer une structure pour la défense et le dernier s’occupe de préparer une belle mise en page.

4.Conclusion

Avez-vous réussi à réaliser l'entièreté du travail demandé ? :

Oui, nous avons réussi à réaliser l'intégralité du travail demandé.

Quelles ont été les principales difficultés rencontrées et comment les avez-vous résolues ? :

La difficulté était sans doute la programmation, le C nécessite une bonne compréhension du langage. Nous avons pu pallier à cela grâce aux cours et en se documentant sur Internet.

Quelles sont vos conclusions personnelles sur votre expérience de travail en groupe et sur les compétences que vous avez développées lors de ce projet ?

Loan : *Je tiens particulièrement à remercier Brice et Samin pour leur travail, je trouve que nous avons formé une très belle équipe et que nous avons fait du bon travail. J'ai apprécié ce projet principalement car je n'avais jamais travaillé sous Linux avant cela, sinon je trouve que l'ensemble du projet est très intéressant.*

Brice : *En travaillant en groupe, j'ai développé des compétences clés en collaboration, communication et gestion des délais. De plus, j'ai pu renforcer mes compétences en programmation, notamment en apprenant le langage C sous Linux, que je n'avais jamais approché avant ma première année à l'EPHEC. Cette expérience a été enrichissante et formatrice.*

Samin : *Je remercie Brice et Loan pour ce beau travail que nous avons réalisés, faire équipe avec eux a été fort agréable. Ce projet m'a également permis d'en apprendre d'avantage sur Linux et de découvrir un nouveau langage de programmation qui est le C. C'était une expérience fort enrichissante.*

5.Annexes

Version fin de partie 1:

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

#define BUFFER_SIZE 1024

int main(int argc, char *argv[]) {
    char buffer[BUFFER_SIZE];
    pid_t pid;
    int status, pipefd[2];

    // Vérifier que la commande a été passée en argument
    if (argc < 2) {
        printf("Usage: %s <command> [arg1] [arg2] ...\n", argv[0]);
        exit(1);
    }

    // Créer un canal de communication entre le processus fils et le processus père
    if (pipe(pipefd) == -1) {
        perror("pipe");
        exit(EXIT_FAILURE);
    }

    // Créer un processus fils
    pid = fork();

    if (pid == -1) {
        printf("Erreur lors de la création du processus fils.\n");
        exit(1);
    } else if (pid == 0) {
        // Nous sommes dans le processus fils

        // Fermer l'extrémité de lecture du canal
        close(pipefd[0]);

        // Rediriger la sortie standard vers l'extrémité d'écriture du canal
        dup2(pipefd[1], STDOUT_FILENO);

        // Exécuter la commande passée en argument
        execvp(argv[1], &argv[1]);

        // Si execvp renvoie, cela signifie qu'il y a eu une erreur
        printf("Erreur lors de l'exécution de la commande.\n");
        exit(1);
    } else {
        // Nous sommes dans le processus père

        // Fermer l'extrémité d'écriture du canal
        close(pipefd[1]);
```

```
    // Lire le résultat de l'exécution de la commande depuis l'extrémité
de lecture du canal
      int nbytes = read(pipefd[0], buffer, BUFFER_SIZE);
// Attendre la fin du processus fils
      wait(&status);
      // Afficher le résultat
      if (nbytes > 0) {
         buffer[nbytes] = '\0'; // Ajouter le caractère de fin de chaîne
         printf("Le processus fils a retourné : %s\n", buffer);
      } else {
         printf("Erreur lors de la lecture du résultat depuis le canal.\n");
      }
       printf("Le processus fils a terminé avec le code de retour %d.\n",
WEXITSTATUS(status));
   }

   return 0;
}
```

Version finale du code :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>

#define BUFFER_SIZE 1024

int main(int argc, char *argv[]) {
   pid_t pid;
   int status, pipefd[2];


   // Vérifier que les fichiers shell ont été passés en argument
   if (argc < 2) {
               printf("Usage:  %s  <shell_script1>  <shell_script2>  ...
<shell_scriptN>\n", argv[0]);
      exit(1);
   }

   // Boucle pour exécuter chaque fichier shell
   for (int i = 1; i < argc; i++) {


   //printf("j'ouvre le fichier\n");
   // Ouvrir le fichier shell
        FILE* shell_file = fopen(argv[i], "r");
        if (!shell_file) {
           perror("Erreur lors de l'ouverture du fichier");
           exit(EXIT_FAILURE);
        }
```

```
        // Créer un canal de communication entre le processus fils et le
processus père
        // crée un canal de communication et vérifie si la création a
réussi
        if (pipe(pipefd) == -1) {
            perror("pipe");
            exit(EXIT_FAILURE);
        }

        // Boucle pour exécuter chaque ligne du fichier shell
        char buffer[BUFFER_SIZE];
        while (fgets(buffer, BUFFER_SIZE, shell_file) != NULL) {
            // Supprimer le caractère de fin de ligne
            buffer[strcspn(buffer, "\n")] = 0;

            // Créer un processus fils pour exécuter la commande
            pid = fork();

            if (pid == -1) {
                printf("Erreur lors de la création du processus fils.\n");
                exit(1);
            } else if (pid == 0) {
                // Nous sommes dans le processus fils
                // printf("Bonjour, je suis le processus fils\n");
                // Fermer l'extrémité de lecture du canal
                close(pipefd[0]);

                // Rediriger la sortie standard vers l'extrémité d'écriture du
canal
                dup2(pipefd[1], STDOUT_FILENO);

                // Exécuter la commande
                // execl("/bin/sh", "sh", "-c", buffer, (char *) NULL);
                char *args[] = {"sh", "-c", buffer, NULL};
                execvp(args[0], args);
                //Si execl renvoie, cela signifie qu'il y a eu une erreur
                //printf("Erreur lors de l'exécution de la commande.\n");
                printf("Erreur lors de l'exécution de la commande.\n");
                exit(1);
            } else {
                // Nous sommes dans le processus père
                // printf("Bonjour, je suis le processus père\n");
                // Fermer l'extrémité d'écriture du canal
                close(pipefd[1]);

                // Lire le résultat de l'exécution de la commande depuis
l'extrémité de lecture du canal
                char result[BUFFER_SIZE];
                int nbytes;
    int total_bytes = 0;
                while ((nbytes = read(pipefd[0], result + total_bytes,
BUFFER_SIZE - total_bytes)) > 0) {
                    total_bytes += nbytes;
                    //printf("Lecture en cours\n");
                }

                // Fermer l'extrémité de lecture du canal
                close(pipefd[0]);
```

```
     // Attendre la fin du processus fils spécifique créé par le fork() actuel
            waitpid(pid, &status, 0);

            // Afficher le résultat
            if (total_bytes >= 0) {
                result[total_bytes] = '\0'; // Ajouter le caractère de fin de chaîne
                 printf("Le processus fils a retourné pour la commande \"%s\" : %s\n", buffer, result);
            } else {
                 printf("Erreur lors de la lecture du résultat depuis le canal.\n");
            }
             printf("Le processus fils a terminé avec le code de retour %d.\n\n\n", WEXITSTATUS(status));
          // condition qui arrête le programme en cas d'erreurs (choix personnel)
          if(WEXITSTATUS(status)!=0){exit(1);}
         }
    }

    // Fermer le fichier shell
    fclose(shell_file);
}
    return 0;
}
```