



Institut National de Statistique et d'Économie Appliquée

Projet de Programmation Système Mini Shell en C

Étudiant : Issam Legssair

Filière : Biostatistique, Démographie et Big Data

Encadrant : Anouar Bouchal

Projet de Programmation Système

Mini Shell en C

17 mai 2025

Table des matières

1	Introduction	2
2	Analyse et conception	2
2.1	Définition d'un Shell	2
2.2	Objectifs fonctionnels	2
2.3	Architecture du programme	2
3	Description du code source	2
3.1	Fonction <code>afficher_prompt()</code>	2
3.2	Fonction <code>lire_commande()</code>	2
3.3	Fonction <code>executer_commande()</code>	2
4	Code source	2
5	Compilation et exécution	3
6	Captures d'écran avec explications	4
6.1	Code Source	4
6.2	Commande <code>ls</code>	7
6.3	Commande <code>pwd</code>	7
6.4	Commande <code>date</code>	8
6.5	Commande <code>exit</code>	8
7	Conclusion	9

1 Introduction

Ce rapport présente un projet réalisé dans le cadre du l'élément de Programmation Système, développé spécifiquement sous environnement Debian. Il s'agit de la création d'un mini shell en C capable d'exécuter des commandes UNIX simples entrées par l'utilisateur.

2 Analyse et conception

2.1 Définition d'un Shell

Un shell est une interface en ligne de commande qui permet à l'utilisateur d'interagir avec le système d'exploitation.

2.2 Objectifs fonctionnels

- Afficher un prompt personnalisé.
- Lire la commande utilisateur.
- Exécuter la commande avec ses arguments.
- Gérer la commande `exit` pour quitter le shell.

2.3 Architecture du programme

- Fonction d'affichage du prompt.
- Lecture et traitement de la commande.
- Exécution dans un processus fils.

3 Description du code source

3.1 Fonction `afficher_prompt()`

Affiche le texte « minishell> » à chaque itération.

3.2 Fonction `lire_commande()`

Lit une ligne entrée par l'utilisateur et supprime le retour à la ligne.

3.3 Fonction `executer_commande()`

- Utilise `fork()` pour créer un processus fils.
- Utilise `execvp()` pour exécuter la commande.
- Attend la fin du processus fils avec `wait()`.

4 Code source

Listing 1 – Code source du mini shell

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

void afficher_prompt() {
```

```

        printf("minishell>_");
    }

    void lire_commande(char* commande) {
        fgets(commande, 1024, stdin);
        commande[strcspn(commande, "\n")] = '\0';
    }

    void executer_commande(char* commande) {
        char* args[64];
        int i = 0;
        args[i] = strtok(commande, "_");
        while (args[i] != NULL) {
            args[++i] = strtok(NULL, "_");
        }

        if (fork() == 0) {
            execvp(args[0], args);
            perror("Erreur_d'ex\ 'ecution");
            exit(EXIT_FAILURE);
        } else {
            wait(NULL);
        }
    }

    int main() {
        char commande[1024];
        while (1) {
            afficher_prompt();
            lire_commande(commande);
            if (strcmp(commande, "exit") == 0)
                break;
            executer_commande(commande);
        }
        return 0;
    }

```

5 Compilation et exécution

Compilation

```
gcc minishell.c -o minishell
```

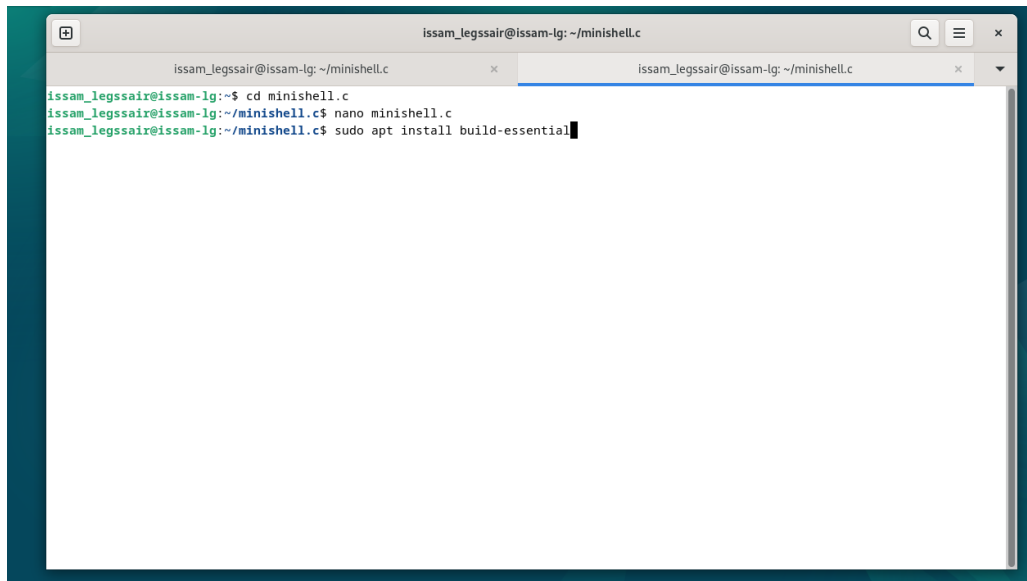


FIGURE 1 – Installation de gcc

Exécution

`./minishell`

6 Captures d'écran avec explications

Dans cette section, nous illustrons différentes commandes testées dans le mini shell. Chaque capture est accompagnée d'une explication du comportement attendu.

6.1 Code Source

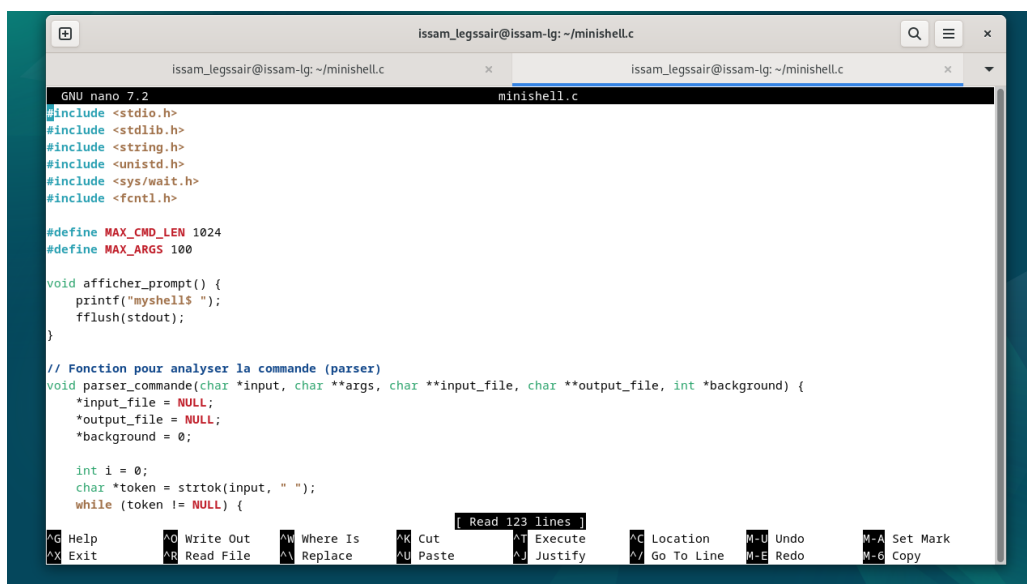
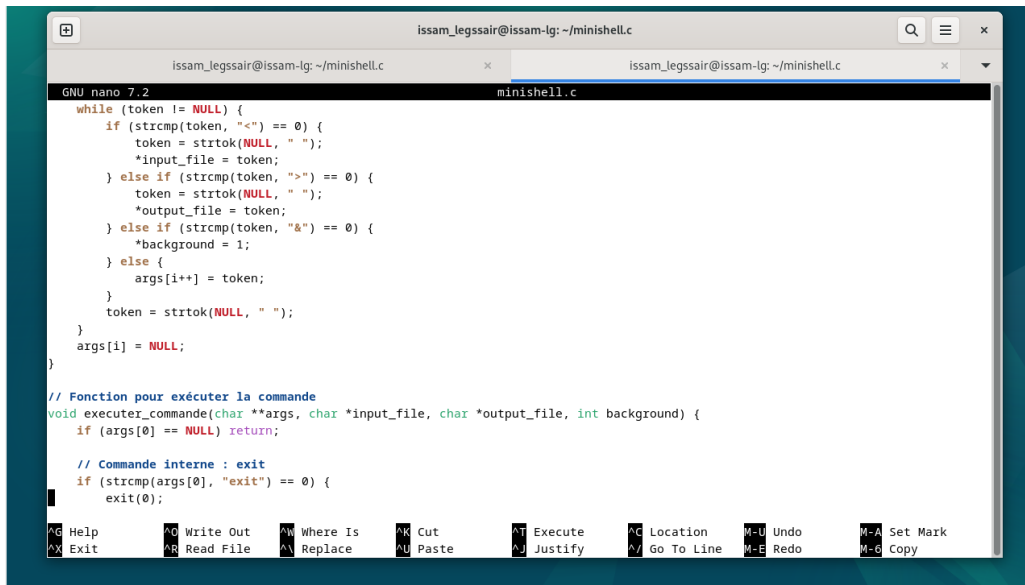


FIGURE 2 – Fonction affiche le prompt minishell>



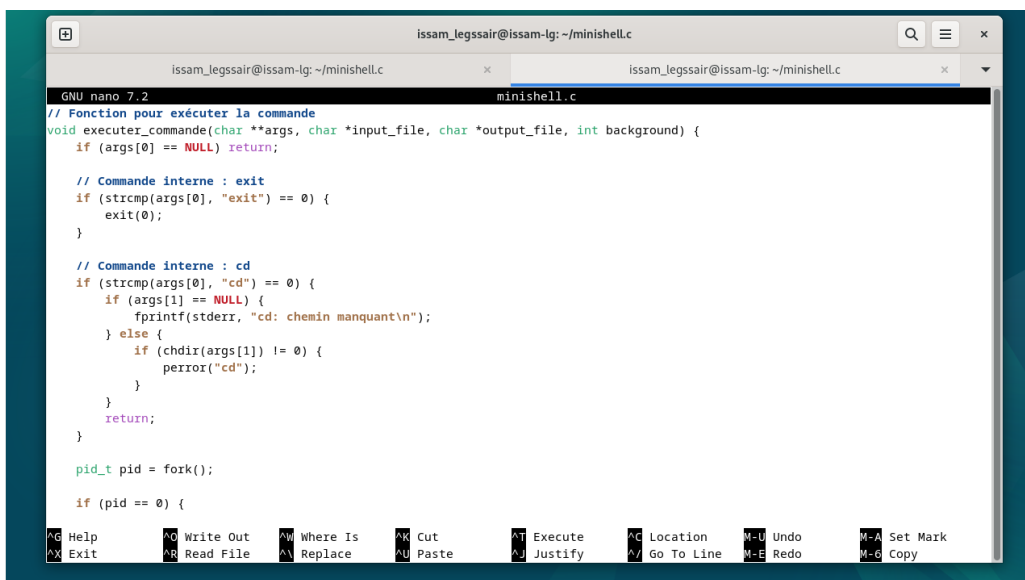
```

issam_legssair@issam-ig: ~/minishell.c
GNU nano 7.2 minishell.c
while (token != NULL) {
    if (strcmp(token, "<") == 0) {
        token = strtok(NULL, " ");
        *input_file = token;
    } else if (strcmp(token, ">") == 0) {
        token = strtok(NULL, " ");
        *output_file = token;
    } else if (strcmp(token, "&") == 0) {
        *background = 1;
    } else {
        args[i++] = token;
    }
    token = strtok(NULL, " ");
}
args[i] = NULL;

// Fonction pour exécuter la commande
void executer_commande(char **args, char *input_file, char *output_file, int background) {
    if (args[0] == NULL) return;

    // Commande interne : exit
    if (strcmp(args[0], "exit") == 0) {
        exit(0);
    }
}
    
```

FIGURE 3 – Suite du code 1



```

issam_legssair@issam-ig: ~/minishell.c
GNU nano 7.2 minishell.c
// Fonction pour exécuter la commande
void executer_commande(char **args, char *input_file, char *output_file, int background) {
    if (args[0] == NULL) return;

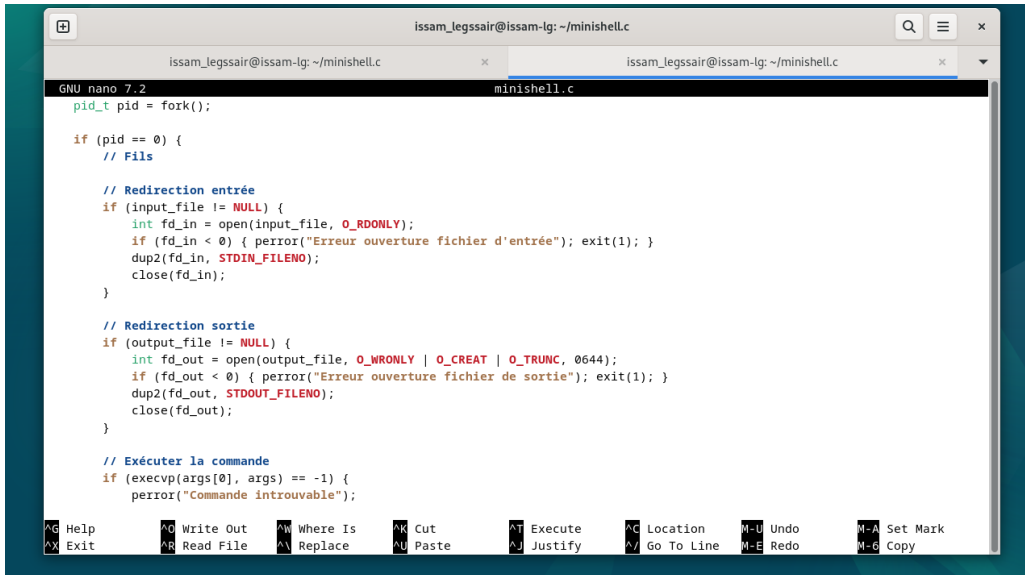
    // Commande interne : exit
    if (strcmp(args[0], "exit") == 0) {
        exit(0);
    }

    // Commande interne : cd
    if (strcmp(args[0], "cd") == 0) {
        if (args[1] == NULL) {
            fprintf(stderr, "cd: chemin manquant\n");
        } else {
            if (chdir(args[1]) != 0) {
                perror("cd");
            }
        }
        return;
    }

    pid_t pid = fork();

    if (pid == 0) {
    }
}
    
```

FIGURE 4 – Suite du code



```

issam_legssair@issam-lg: ~/minishell.c
GNU nano 7.2 minishell.c
pid_t pid = fork();

if (pid == 0) {
    // Fils

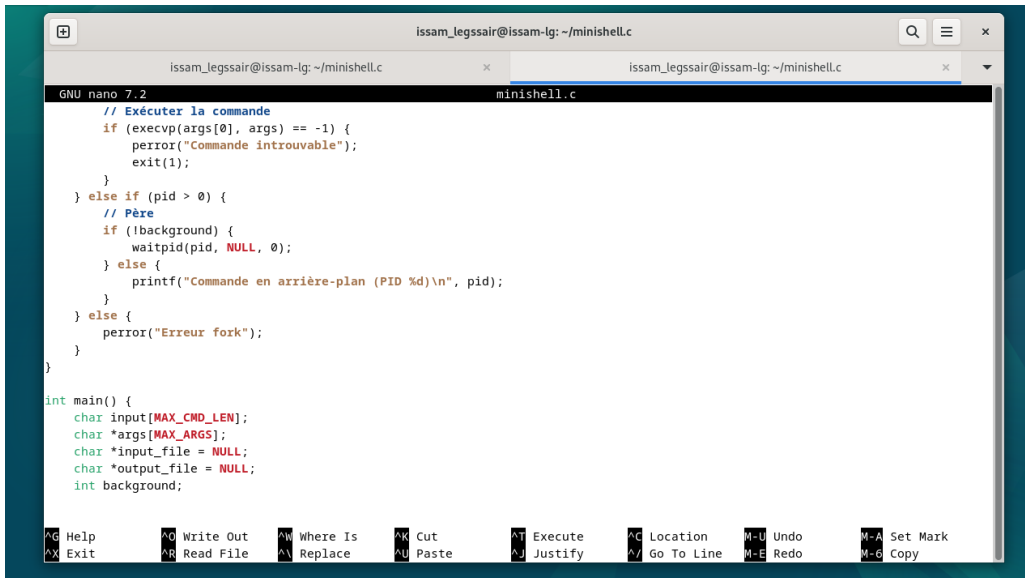
    // Redirection entrée
    if (input_file != NULL) {
        int fd_in = open(input_file, O_RDONLY);
        if (fd_in < 0) { perror("Erreur ouverture fichier d'entrée"); exit(1); }
        dup2(fd_in, STDIN_FILENO);
        close(fd_in);
    }

    // Redirection sortie
    if (output_file != NULL) {
        int fd_out = open(output_file, O_WRONLY | O_CREAT | O_TRUNC, 0644);
        if (fd_out < 0) { perror("Erreur ouverture fichier de sortie"); exit(1); }
        dup2(fd_out, STDOUT_FILENO);
        close(fd_out);
    }

    // Exécuter la commande
    if (execvp(args[0], args) == -1) {
        perror("Commande introuvable");
    }
}

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   ^U Undo       ^M-A Set Mark
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line  ^M-E Redo     ^M-G Copy
    
```

FIGURE 5 – Suite de code



```

issam_legssair@issam-lg: ~/minishell.c
GNU nano 7.2 minishell.c

    // Exécuter la commande
    if (execvp(args[0], args) == -1) {
        perror("Commande introuvable");
        exit(1);
    }
} else if (pid > 0) {
    // Père
    if (!background) {
        waitpid(pid, NULL, 0);
    } else {
        printf("Commande en arrière-plan (PID %d)\n", pid);
    }
} else {
    perror("Erreur fork");
}
}

int main() {
    char input[MAX_CMD_LEN];
    char *args[MAX_ARGS];
    char *input_file = NULL;
    char *output_file = NULL;
    int background;
}
    
```

FIGURE 6 – Suite du code

```

issam_legssair@issam-lg: ~/minishell.c
GNU nano 7.2 minishell.c
char input[MAX_CMD_LEN];
char *args[MAX_ARGS];
char *input_file = NULL;
char *output_file = NULL;
int background;

while (1) {
    afficher_prompt();

    if (fgets(input, MAX_CMD_LEN, stdin) == NULL) {
        break;
    }

    // Supprimer le saut de ligne
    input[strcspn(input, "\n")] = '\0';

    parser_commande(input, args, &input_file, &output_file, &background);
    executer_commande(args, input_file, output_file, background);
}

return 0;
}

```

FIGURE 7 – Fin du code

6.2 Commande ls

But : Lister le contenu du répertoire courant.

```

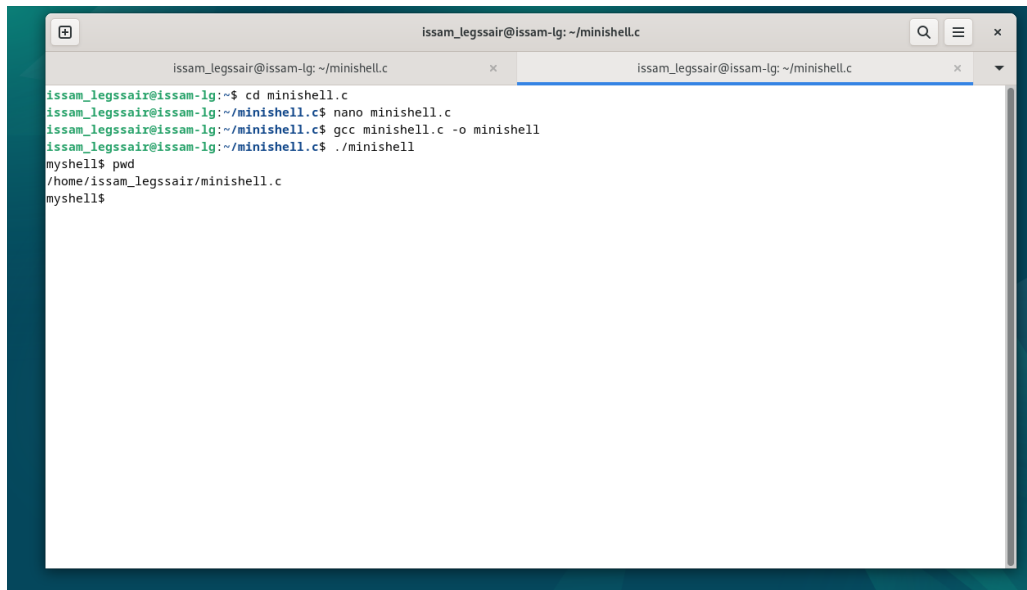
issam_legssair@issam-lg: ~/minishell.c
issam_legssair@issam-lg:~$ ls
Desktop Documents Downloads minishell.c Music Pictures Public Security_Manager.sh Security_Manager.zip Templates Videos
issam_legssair@issam-lg:~$ cd minishell.c
issam_legssair@issam-lg:~/minishell.c$ nano minishell.c
issam_legssair@issam-lg:~/minishell.c$ gcc minishell.c -o minishell
issam_legssair@issam-lg:~/minishell.c$ ./minishell
mysHELL$ ls
fichier.txt minishell minishell.c minishell.c.save minishell.c.save.1 minishell.c.save.2
mysHELL$

```

FIGURE 8 – Commande ls affichant les fichiers du répertoire courant

6.3 Commande pwd

But : Afficher le chemin absolu du répertoire courant.



```

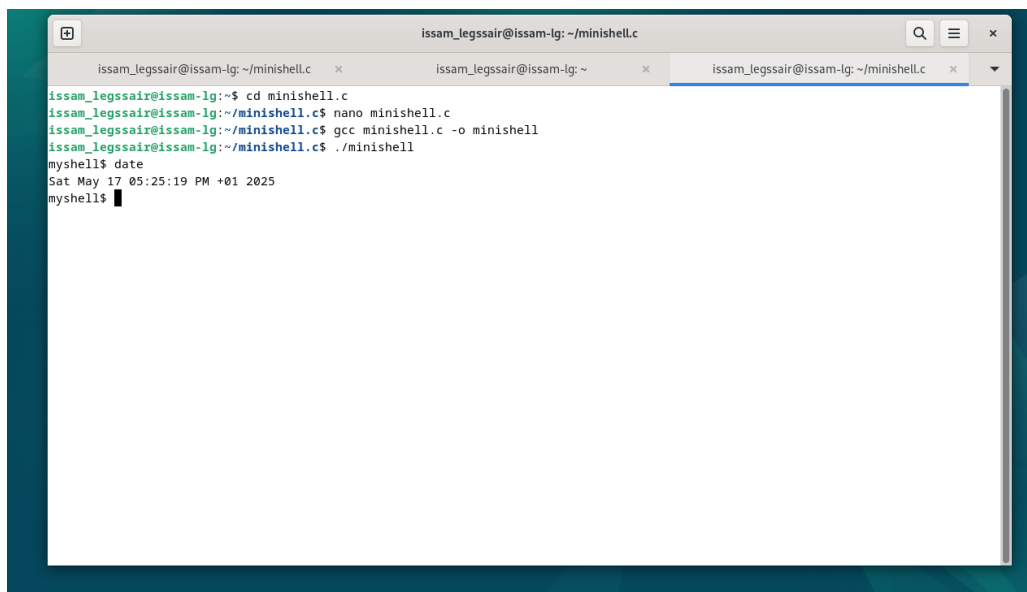
issam_legssair@issam-ig: ~/minishell.c
issam_legssair@issam-ig:~$ cd minishell.c
issam_legssair@issam-ig:~/minishell.c$ nano minishell.c
issam_legssair@issam-ig:~/minishell.c$ gcc minishell.c -o minishell
issam_legssair@issam-ig:~/minishell.c$ ./minishell
myshell$ pwd
/home/issam_legssair/minishell.c
myshell$

```

FIGURE 9 – Commande `pwd` pour afficher le chemin du répertoire

6.4 Commande `date`

But : Afficher la date et l'heure actuelles.



```

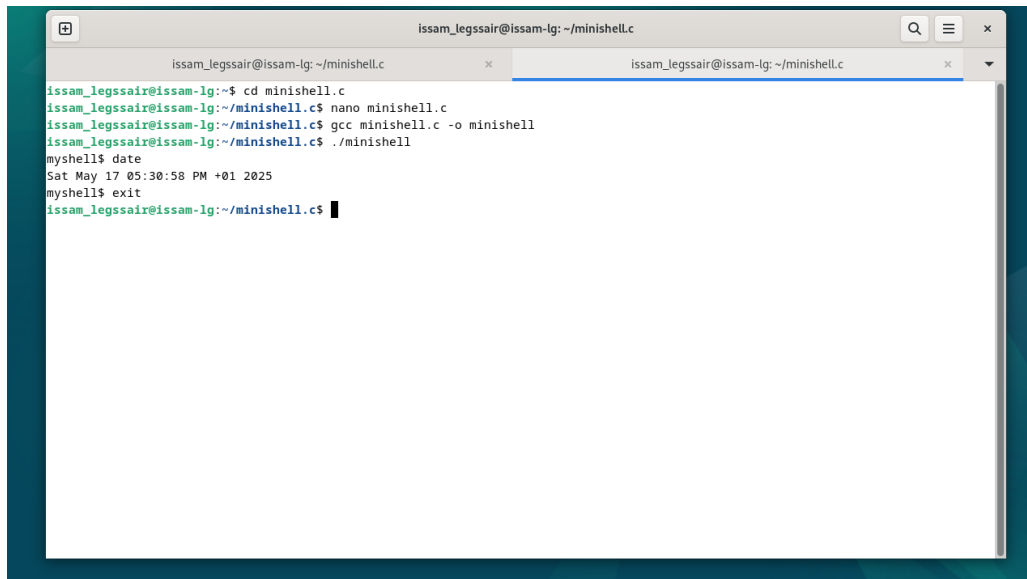
issam_legssair@issam-ig: ~/minishell.c
issam_legssair@issam-ig:~$ cd minishell.c
issam_legssair@issam-ig:~/minishell.c$ nano minishell.c
issam_legssair@issam-ig:~/minishell.c$ gcc minishell.c -o minishell
issam_legssair@issam-ig:~/minishell.c$ ./minishell
myshell$ date
Sat May 17 05:25:19 PM +01 2025
myshell$

```

FIGURE 10 – Commande `date` pour voir la date système

6.5 Commande `exit`

But : Quitter le shell proprement.



```

issam_legssair@issam-lg: ~/minishell.c
issam_legssair@issam-lg:~$ cd minishell.c
issam_legssair@issam-lg:~/minishell.c$ nano minishell.c
issam_legssair@issam-lg:~/minishell.c$ gcc minishell.c -o minishell
issam_legssair@issam-lg:~/minishell.c$ ./minishell
myshell$ date
Sat May 17 05:30:58 PM +01 2025
myshell$ exit
issam_legssair@issam-lg:~/minishell.c$

```

FIGURE 11 – Commande `exit` pour quitter le shell

7 Conclusion

Ce projet nous a permis de mieux comprendre les mécanismes internes d'un shell Unix. L'utilisation des appels système comme `fork` et `execvp` a été essentielle pour simuler un fonctionnement réel.