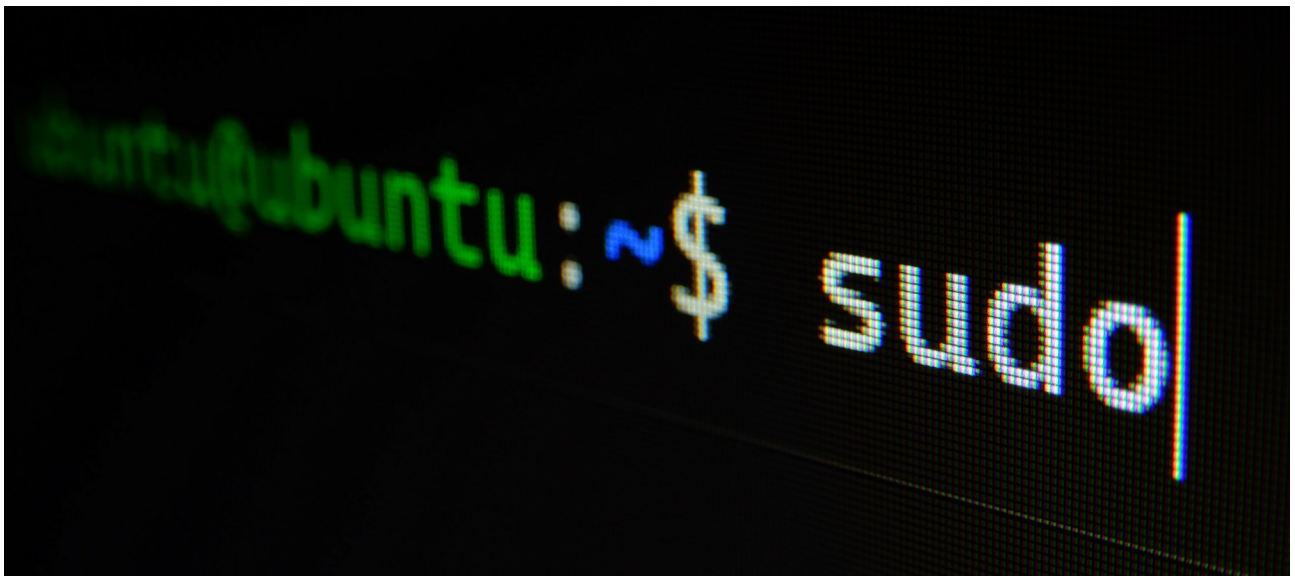




# PROJET DE FIN DE MODULE: MONCEFSHELL

DATE: 15/05/2023



RÉALISÉ PAR : BEN ISSA MONCEF  
ENCADRÉ PAR: EL AMRANI CHAKER

## Introduction:

Le projet "MoncefSHELL" consiste en la création d'un shell personnalisé en langage C. Un shell est un programme qui permet à l'utilisateur d'interagir avec le système d'exploitation en utilisant une interface en ligne de commande. Dans ce projet, nous avons développé un shell basé sur les fonctionnalités essentielles d'un shell standard, telles que l'exécution de commandes, la gestion des processus et la manipulation des fichiers.

## Tableau des commandes:

Cammande	Syntaxe	Fonction exécutée	Description
lister	lister	lister()	Affiche la liste des fichiers dans le répertoire courant.
ouvrir	ouvrir nom_fichier	ouvrir()	Ouvre un fichier spécifié.
manuel	manuel	manuel()	Liste les commandes avec une description.
creer	creer nom_fichier	creer()	Crée un nouveau fichier.
supprimer	supprimer nom_fichier	supprimer()	Supprime un fichier spécifié
renommer	renommer fichier nex_name	renommer()	Renomme un fichier spécifié
calculer	calculer op num1 num2	calculer()	Effectue un calcul spécifié
compter	compter	compter()	Compte le nombre de mots dans un fichier spécifié
envoyer	envoyer	envoyer()	Envoie un message spécifié via une file de messages
recevoir	recevoir	recevoir()	Reçoit un message via une file de messages.
envoyerf	envoyerf	envoyerf()	Envoie un fichier spécifié via un tube FIFO
recevoirf	recevoirf	recevoirf()	Reçoit un fichier spécifié via un tube FIFO
quitter	quitter	-	Quitte le programme

## Explication du code source:

### Bibliothèques ajoutées:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <semaphore.h>
#include <sys/ipc.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <pthread.h>
#include <sys/msg.h>
```

### Définir des constantes:

```
#define BUFFER_SIZE 1024
#define SEM_NAME "/my_semaphore"
#define MESSAGE_SIZE 1024
#define FIFO_PATH "/tmp/mon_fifo"
```

### Déclarations de fonctions:

```
void creer(char **args, int arg_count);
void lister(char **args, int arg_count);
void supprimer(char **args, int arg_count);
void renommer(char **args);
void *addition(void *arg);
void *soustraction(void *arg);
void *multiplication(void *arg);
void calculer(char **args, int arg_count);
void compter(char **args);
void ouvrir(char **args, int arg_count);
int envoyer_message(int file_id, long type, char *message_data);
void recevoir_message(int cle, int type);
void recevoirf(char **args, int arg_count);
void envoyerf(char **args, int arg_count);
void manuel();
```

## La fonction main():

Le shell affiche une invite de commande "MoncefSHELL >> " et attend que l'utilisateur saisisse une commande. Les commandes peuvent être des opérations de base telles que lister, ouvrir, créer, supprimer, renommer des fichiers, ainsi que des opérations plus spécifiques telles que calculer ou compter. Le shell prend également en charge l'envoi et la réception de messages.

Si la commande saisie correspond à l'une des commandes prises en charge, le shell exécute la fonction appropriée pour traiter cette commande. Sinon, le shell crée un processus fils pour exécuter la commande externe en utilisant la fonction `execvp`.

Le boucle principale du programme continue de lire et de traiter les commandes tant que l'utilisateur n'entre pas la commande "quitter". Lorsque l'utilisateur quitte le shell, le programme ferme le sémaphore, supprime le tube nommé et se termine.

```
int main(){

    signal(SIGINT,SIG_IGN);

    int cle = 1234;
    char input[256];
    char buffer[BUFFER_SIZE];
    char *args[100];
    int arg_count;
    mkfifo(FIFO_PATH, 0666);

    sem_t *my_semaphore = sem_open(SEM_NAME, O_CREAT, 0644, 1);
    if (my_semaphore == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }

    while (1) {
        printf("MoncefSHELL >> ");
        fgets(buffer, BUFFER_SIZE, stdin);

        arg_count = 0;
        args[arg_count] = strtok(buffer, " \n");
        while (args[arg_count] != NULL) {
            arg_count++;
            args[arg_count] = strtok(NULL, " \n");
        }

        if (arg_count == 0) {
            continue;
        }

        if (strcmp(args[0], "lister") == 0) {
            lister(args, arg_count);
        }
        else if (strcmp(args[0], "ouvrir") == 0) {
```

```

        ouvrir(args, arg_count);
    }
    else if (strcmp(args[0], "manuel") == 0) {
        manuel();
    }
    else if (strcmp(args[0], "creer") == 0) {
        creer(args, arg_count);
    }
    else if (strcmp(args[0], "supprimer") == 0) {
        supprimer(args, arg_count);
    }
    else if (strcmp(args[0], "renommer") == 0) {
        renommer(args);
    }
    else if (strcmp(args[0], "calculer") == 0) {
        calculer(args, arg_count);
    }
    else if (strcmp(args[0], "compter") == 0) {
        compter(args);
    }
    else if (strcmp(args[0], "envoyer") == 0) {
        char message[MESSAGE_SIZE];
        printf("Entrez le message à envoyer : ");
        scanf("%s", message);
        envoyer_message(msgget(cle, 0666 | IPC_CREAT), 1, message);
    }
    else if (strcmp(args[0], "recevoir") == 0) {
        recevoir_message(cle, 1);
    }
    else if (strcmp(args[0], "envoyerf") == 0) {
        envoyerf(args, arg_count);
    }
    else if (strcmp(args[0], "recevoirf") == 0) {
        recevoirf(args, arg_count);
    }

    else if (strcmp(args[0], "quitter") == 0) {
        sem_close(my_semaphore);
        sem_unlink(SEM_NAME);
        exit(0);
    }
    else {
        pid_t pid = fork();

        if (pid == 0) {
            execvp(args[0], args);
            printf("Erreur : la commande '%s' a échoué\n", args[0]);
            exit(1);
        } else if (pid < 0) {
            printf("Erreur : échec de la création d'un processus fils\n");
        } else {

```

```

        waitpid(pid, NULL, 0);
    }
}

unlink(FIFO_PATH);
return 0;
}

```

## Fonction "creer" : Crée un fichier:

La fonction `creer` est utilisée dans le programme shell pour créer un nouveau fichier en utilisant la commande système `"touch"`. Elle vérifie d'abord le nombre d'arguments, puis crée un processus fils. Le processus fils exécute la commande `"touch"` avec le nom du fichier à créer, tandis que le processus parent attend la fin de l'exécution. Si la commande échoue, un message d'erreur est affiché.

```
void creer(char **args, int arg_count) {
    if (arg_count != 2) {
        printf("Erreur : nombre d'arguments incorrect pour la commande 'creer'\n");
        return;
    }

    pid_t pid = fork();

    if (pid == 0) {
        execlp("touch", "touch", args[1], NULL);
        printf("Erreur : la commande 'touch' a échoué\n");
        exit(1);
    } else if (pid < 0) {
        printf("Erreur : échec de la création d'un processus fils\n");
    } else {
        waitpid(pid, NULL, 0);
    }
}
```

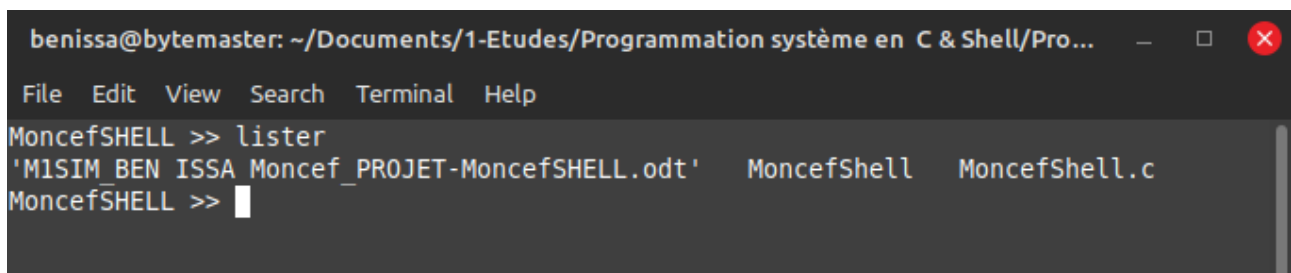
[illegible]

## Fonction "lister" : Affiche la liste des fichiers:

La fonction lister est utilisée dans le programme shell pour afficher le contenu d'un répertoire en utilisant la commande système "ls".

Elle crée un processus fils en utilisant la fonction fork. Le processus fils utilise ensuite la fonction execvp pour exécuter la commande "ls" avec les arguments fournis. Cela permet d'afficher le contenu du répertoire spécifié.

Si l'exécution de la commande "ls" échoue, un message d'erreur est affiché. Sinon, le processus parent attend que le processus fils se termine en utilisant la fonction waitpid.



```
benissa@bytemaster: ~/Documents/1-Etudes/Programmation système en C & Shell/Pro...
File Edit View Search Terminal Help
MoncefSHELL >> lister
'MISIM BEN ISSA Moncef_PROJET-MoncefSHELL.odt'  MoncefShell  MoncefShell.c
MoncefSHELL >> 
```

```
void lister(char **args, int arg_count) {
    pid_t pid = fork();

    if (pid == 0) {
        execvp("ls", args);
        printf("Erreur : la commande 'ls' a échoué\n");
        exit(1);
    } else if (pid < 0) {
        printf("Erreur : échec de la création d'un processus fils\n");
    } else {
        waitpid(pid, NULL, 0);
    }
}
```

## Fonction "supprimer" : Supprime un fichier spécifié:

La fonction supprimer est utilisée dans le programme shell pour supprimer un fichier en utilisant la commande système "rm". Elle vérifie d'abord le nombre d'arguments, puis crée un processus fils. Le processus fils exécute la commande "rm" avec le nom du fichier à supprimer, tandis que le processus parent attend la fin de l'exécution. Si la commande échoue, un message d'erreur est affiché.

```
void supprimer(char **args, int arg_count) {
    if (arg_count != 2) {
        printf("Erreur : nombre d'arguments incorrect pour la commande 'supprimer'\n");
        return;
    }

    pid_t pid = fork();
```

```

if (pid == 0) {
    execlp("rm", "rm", args[1], NULL);
    printf("Erreur : la commande 'rm' a échoué\n");
    exit(1);
} else if (pid < 0) {
    printf("Erreur : échec de la création d'un processus fils\n");
} else {
    waitpid(pid, NULL, 0);
}
}

```

```

MISIM BEN ISSA Moncef_PROJET-MoncefSHELL.odt MoncefShell.c
MoncefSHELL >> supprimer fichier
MoncefSHELL >> lister
'MISIM BEN ISSA Moncef_PROJET-MoncefSHELL.odt' MoncefShell MoncefShell.c
MoncefSHELL >>

```

## Fonction "renommer" : Renomme un fichier spécifié:

La fonction renommer est utilisée dans le programme shell pour renommer un fichier en utilisant la commande système "mv". Elle compte d'abord le nombre d'arguments passés. Si le nombre d'arguments n'est pas égal à 3, un message d'erreur est affiché. Ensuite, elle crée un processus fils qui exécute la commande "mv" avec les arguments pour renommer le fichier. Le processus parent attend la fin de l'exécution.

```

void renommer(char **args) {
    int arg_count = 0;
    while (args[arg_count] != NULL) {
        arg_count++;
    }

    if (arg_count != 3) {
        printf("Erreur : nombre d'arguments incorrect pour la commande 'renommer'\n");
        return;
    }

    pid_t pid = fork();

    if (pid == 0) {
        execlp("mv", "mv", args[1], args[2], NULL);
        printf("Erreur : la commande 'mv' a échoué\n");
        exit(1);
    } else if (pid < 0) {
        printf("Erreur : échec de la création d'un processus fils\n");
    } else {
        waitpid(pid, NULL, 0);
    }
}

```



```
MoncefSHELL >> renommer test fichier
MoncefSHELL >> lister
fichier                               MoncefShell
'MISIM BEN ISSA Moncef_PROJET-MoncefSHELL.odt' MoncefShell.c
MoncefSHELL >> █
```

## Fonction "addition" : Effectue une addition de deux nombres:

```
void *addition(void *arg) {
    int *args = (int *) arg;
    int a = args[0];
    int b = args[1];
    int *result = (int *) malloc(sizeof(int));
    *result = a + b;
    pthread_exit(result);
}
```

## Fonction "soustraction" : Effectue une soustraction de deux nombres:

```
void *soustraction(void *arg) {
    int *args = (int *) arg;
    int a = args[0];
    int b = args[1];
    int *result = (int *) malloc(sizeof(int));
    *result = a - b;
    pthread_exit(result);
}
```

## Fonction "multiplication" : Effectue une multiplication de deux nombres:

```
void *multiplication(void *arg) {
    int *args = (int *) arg;
    int a = args[0];
    int b = args[1];
    int *result = (int *) malloc(sizeof(int));
    *result = a * b;
    pthread_exit(result);
}
```

## Fonction "calculer" : Effectue un calcul spécifié:

La fonction calculer est utilisée dans le programme shell pour effectuer des opérations mathématiques sur deux entiers. Elle vérifie d'abord si le nombre d'arguments est correct (4

arguments : le nom de la commande, l'opération, le premier entier et le deuxième entier). Si le nombre d'arguments est incorrect, un message d'erreur est affiché.

Ensuite, la fonction extrait l'opération et les entiers à partir des arguments passés. Elle crée un thread en fonction de l'opération (addition, soustraction ou multiplication) et les arguments sont passés au thread.

Le thread est attendu avec `pthread_join` et le résultat est affiché. Enfin, la mémoire allouée pour le résultat est libérée.

```
void calculer(char **args, int arg_count) {
    if (arg_count != 4) {
        printf("Erreur : nombre d'arguments incorrect pour la commande 'calculer'\n");
        return;
    }

    char *operation = args[1];
    int a = atoi(args[2]);
    int b = atoi(args[3]);

    pthread_t thread;
    void *result;

    if (strcmp(operation, "+") == 0) {
        int arguments[2] = {a, b};
        pthread_create(&thread, NULL, addition, (void *) arguments);
    } else if (strcmp(operation, "-") == 0) {
        int arguments[2] = {a, b};
        pthread_create(&thread, NULL, soustraction, (void *) arguments);
    } else if (strcmp(operation, "*") == 0) {
        int arguments[2] = {a, b};
        pthread_create(&thread, NULL, multiplication, (void *) arguments);
    } else {
        printf("Erreur : opération non valide pour la commande 'calculer'\n");
        return;
    }

    pthread_join(thread, &result);

    printf("%d %s %d = %d\n", a, operation, b, *((int *) result));
    free(result);
}
```

```
MoncefSHELL >> calculer + 3 4
3 + 4 = 7
MoncefSHELL >> calculer - 7 8
7 - 8 = -1
MoncefSHELL >> calculer * 9 10
9 * 10 = 90
MoncefSHELL >> █
```

## Fonction "compter" : Compte le nombre de mots dans un fichier spécifié:

La fonction compter est utilisée dans le programme shell pour compter le nombre de lignes dans la sortie de la commande "ls". Elle crée un tube anonyme avec pipe pour établir une communication entre le processus parent et le processus fils.

Dans le processus fils, il ferme l'extrémité d'écriture du tube, redirige la sortie standard vers l'entrée du tube à l'aide de dup2, puis exécute la commande "wc -l" pour compter les lignes à l'aide de execlp. En cas d'erreur, un message est affiché.

Dans le processus parent, il ferme l'extrémité de lecture du tube. La sortie de la commande "ls" est redirigée vers l'entrée du tube en utilisant popen et les lignes sont écrites dans le tube à l'aide de write. Ensuite, il ferme l'extrémité d'écriture du tube et attend la fin du processus fils avec wait.

La fonction permet ainsi de compter le nombre de fichiers/lignes générés par la commande "ls".

```
void compter(char **args) {
    int fd[2];
    pid_t pid;
    char buffer[BUFFER_SIZE];

    if (pipe(fd) == -1) {
        printf("Erreur : échec de la création d'un tube anonyme\n");
        return;
    }

    pid = fork();
    if (pid < 0) {
        printf("Erreur : échec de la création d'un processus fils\n");
        return;
    }

    if (pid == 0) {
        // Code du fils
        close(fd[1]); // Fermer l'extrémité d'écriture du tube

        // Rediriger la sortie standard vers l'entrée du tube
        dup2(fd[0], STDIN_FILENO);
```

```

// Exécuter la commande "wc -l" pour compter les lignes
execlp("wc", "wc", "-l", NULL);

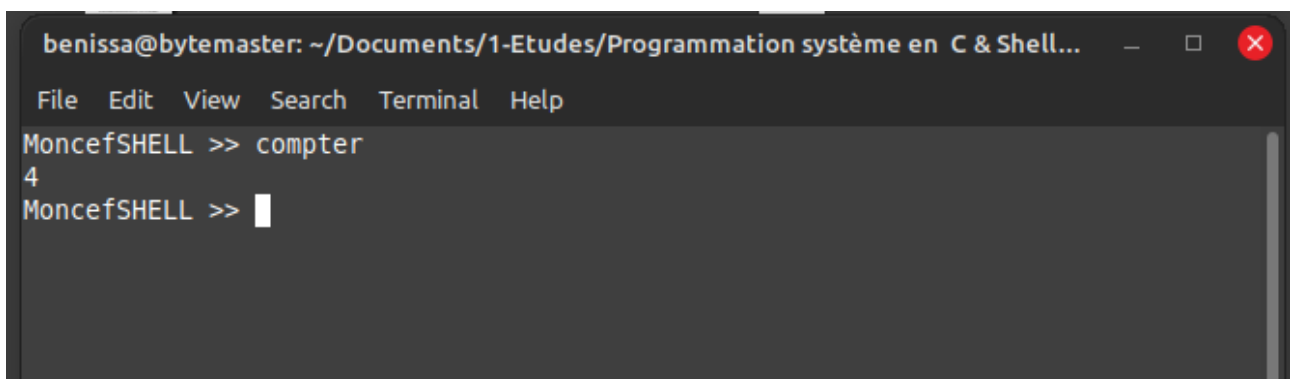
// En cas d'erreur
printf("Erreur : échec de l'exécution de la commande 'wc'\n");
exit(1);
} else {
// Code du parent
close(fd[0]); // Fermer l'extrémité de lecture du tube

// Rediriger la sortie de la commande "ls" vers l'entrée du tube
FILE *fp = popen("ls", "r");
while (fgets(buffer, BUFFER_SIZE, fp) != NULL) {
    write(fd[1], buffer, strlen(buffer));
}
pclose(fp);

close(fd[1]); // Fermer l'extrémité d'écriture du tube

// Attendre la fin du processus fils
wait(NULL);
}
}

```



The screenshot shows a terminal window titled "benissa@bytemaster: ~/Documents/1-Etudes/Programmation système en C & Shell...". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The prompt is "MoncefSHELL >>". The user has entered the command "compter", and the terminal has outputted the number "4". The prompt is now "MoncefSHELL >> " followed by a cursor.

## Fonction "ouvrir" : Ouvre un fichier spécifié:

La fonction ouvrir du programme shell permet d'ouvrir un fichier ou une URL en utilisant la commande système "xdg-open". Elle vérifie d'abord le nombre d'arguments et affiche un message d'erreur en cas d'erreur. Ensuite, elle crée un sémaphore pour assurer l'accès exclusif à la ressource partagée. Elle attend ensuite l'autorisation d'accéder à la ressource avec `sem_wait`. Un processus fils est créé pour exécuter la commande "xdg-open", et le processus parent attend la fin de l'exécution avec `waitpid`. Enfin, l'accès à la ressource partagée est libéré avec `sem_post` et le sémaphore est fermé avec `sem_close`. Cette fonction assure ainsi un accès contrôlé lors de l'ouverture de fichiers ou d'URLs.

```

void ouvrir(char **args, int arg_count) {
    if (arg_count != 2) {
        printf("Erreur : nombre d'arguments incorrect pour la commande 'ouvrir'\n");
        return;
    }

    sem_t *my_semaphore = sem_open(SEM_NAME, O_CREAT, 0644, 1);
    if (my_semaphore == SEM_FAILED) {
        perror("sem_open");
        exit(EXIT_FAILURE);
    }

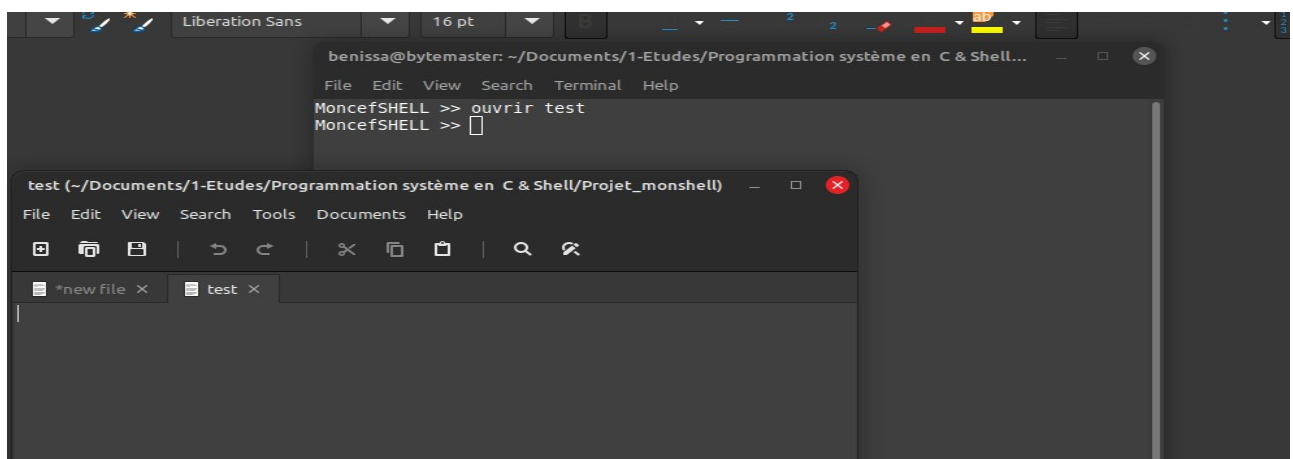
    sem_wait(my_semaphore);

    pid_t pid = fork();

    if (pid == 0) {
        execlp("xdg-open", "xdg-open", args[1], NULL);
        printf("Erreur : la commande 'xdg-open' a échoué\n");
        exit(1);
    } else if (pid < 0) {
        printf("Erreur : échec de la création d'un processus fils\n");
    } else {
        waitpid(pid, NULL, 0);
        sem_post(my_semaphore);
    }

    sem_close(my_semaphore);
}

```



## Fonction "envoyer\_message" : Envoie un message via une file de messages:

La fonction envoyer\_message envoie un message avec le type et les données spécifiés à une file de messages identifiée par file\_id. Si l'envoi est réussi, elle affiche un message de succès, sinon elle affiche une erreur.

```
int envoyer_message(int file_id, long type, char *message_data) {
    struct message msg;
    msg.type = type;
    strncpy(msg.data, message_data, MESSAGE_SIZE);

    int result = msgsnd(file_id, &msg, strlen(msg.data)+1, IPC_NOWAIT);
    if (result == -1) {
        perror("Erreur : impossible d'envoyer le message");
        return -1;
    }

    printf("Message envoyé avec succès : %s\n", message_data);
    return 0;
}
```

```
MoncefSHELL >> envoyer
Entrez le message à envoyer : mon_msg
Message envoyé avec succès : mon_msg
MoncefSHELL >> MoncefSHELL >> □
```

## Fonction "recevoir\_message" : Reçoit un message via une file de messages:

La fonction recevoir\_message ouvre une file de messages identifiée par cle et reçoit un message du type spécifié. Elle affiche le contenu du message reçu et ferme ensuite la file de messages. En cas d'erreur lors de l'ouverture, la réception ou la fermeture de la file, elle affiche un message d'erreur.

```
void recevoir_message(int cle, int type) {
    // Ouvrir la file de messages
    int file = msgget(cle, 0666 | IPC_CREAT);
    if (file == -1) {
        perror("Erreur lors de l'ouverture de la file de messages");
        exit(EXIT_FAILURE);
    }

    // Recevoir un message du type spécifié
    struct message msg;
    int res = msgrcv(file, &msg, MESSAGE_SIZE, type, 0);
}
```

```

if (res == -1) {
    perror("Erreur lors de la réception du message");
    exit(EXIT_FAILURE);
}

// Afficher le message reçu
printf("Message reçu : %s\n", msg.data);

// Fermer la file de messages
res = msgctl(file, IPC_RMID, NULL);
if (res == -1) {
    perror("Erreur lors de la fermeture de la file de messages");
    exit(EXIT_FAILURE);
}
}

```

```

MoncefSHELL >> recevoir
Message reçu : mon_msg
MoncefSHELL >> 

```

## Fonction "recevoirf" : Reçoit un fichier via un tube FIFO:

La fonction recevoirf vérifie le nombre d'arguments reçus et ouvre le fichier FIFO spécifié en mode lecture. Elle lit ensuite le contenu du message depuis le fichier FIFO, ferme le fichier et affiche le message reçu. En cas d'erreur lors de l'ouverture du fichier ou de la lecture du message, elle affiche un message d'erreur correspondant.

```

void recevoirf(char **args, int arg_count) {
    if (arg_count != 2) {
        printf("Erreur : nombre d'arguments incorrect pour la commande 'recevoirf'\n");
        return;
    }

    // Ouvrir le fichier FIFO en lecture
    FILE *fifo = fopen(args[1], "r");
    if (fifo == NULL) {
        printf("Erreur : impossible d'ouvrir le fichier FIFO '%s'\n", args[1]);
        return;
    }

    char message[MESSAGE_SIZE];

    // Lire le message depuis le fichier FIFO

```

```
fscanf(fifo, "%s", message);
fclose(fifo);

printf("Message reçu : %s\n", message);
}
```

```
MoncefSHELL >> recevoirf fifo
Message reçu : moncef
MoncefSHELL >> █
```

## Fonction "envoyerf" : Envoie un fichier via un tube FIFO:

La fonction envoyerf vérifie le nombre d'arguments reçus et ouvre le fichier FIFO spécifié en mode écriture. Elle demande ensuite à l'utilisateur d'entrer le message à envoyer, puis elle écrit ce message dans le fichier FIFO. Après avoir fermé le fichier, elle affiche un message de confirmation indiquant que le message a été envoyé avec succès. En cas d'erreur lors de l'ouverture du fichier ou de l'écriture du message, elle affiche un message d'erreur correspondant.

```
void envoyerf(char **args, int arg_count) {
    if (arg_count != 2) {
        printf("Erreur : nombre d'arguments incorrect pour la commande 'envoyerf'\n");
        return;
    }

    // Ouvrir le fichier FIFO en écriture
    FILE *fifo = fopen(args[1], "w");
    if (fifo == NULL) {
        printf("Erreur : impossible d'ouvrir le fichier FIFO '%s'\n", args[1]);
        return;
    }

    char message[MESSAGE_SIZE];
    printf("Entrez le message à envoyer : ");
    scanf("%s", message);

    // Écrire le message dans le fichier FIFO
    fprintf(fifo, "%s", message);
    fclose(fifo);

    printf("Message envoyé avec succès : %s\n", message);
}
```



```
MoncefSHELL >> envoyerf fifo
Entrez le message à envoyer : moncef
Message envoyé avec succès : moncef
MoncefSHELL >> MoncefSHELL >> 
```

## Fonction "manuel" : Affiche la liste des commandes disponibles:

La fonction manuel affiche la liste des commandes disponibles avec leurs descriptions dans le programme. Cela permet aux utilisateurs de connaître les fonctionnalités offertes.

```
void manuel() {
    printf("Liste des commandes :\n");
    printf("lister\t\tAffiche la liste des fichiers dans le répertoire courant.\n");
    printf("ouvrir\t\tOuvre un fichier spécifié.\n");
    printf("creer\t\tCrée un nouveau fichier.\n");
    printf("supprimer\tSupprime un fichier spécifié.\n");
    printf("renommer\tRenomme un fichier spécifié.\n");
    printf("calculer\tEffectue un calcul spécifié.\n");
    printf("compter\t\tCompte le nombre de mots dans un fichier spécifié.\n");
    printf("envoyer\t\tEnvoie un message spécifié via une file de messages.\n");
    printf("recevoir\tReçoit un message via une file de messages.\n");
    printf("envoyerf\tEnvoie un fichier spécifié via un tube FIFO.\n");
    printf("recevoirf\tReçoit un fichier spécifié via un tube FIFO.\n");
    printf("quitter\t\tQuitte le programme.\n");
}
```

```
MoncefSHELL >> manuel
Liste des commandes :
lister          Affiche la liste des fichiers dans le répertoire courant.
ouvrir          Ouvre un fichier spécifié.
creer           Crée un nouveau fichier.
supprimer       Supprime un fichier spécifié.
renommer        Renomme un fichier spécifié.
calculer        Effectue un calcul spécifié.
compter         Compte le nombre de mots dans un fichier spécifié.
envoyer         Envoie un message spécifié via une file de messages.
recevoir        Reçoit un message via une file de messages.
envoyerf        Envoie un fichier spécifié via un tube FIFO.
recevoirf       Reçoit un fichier spécifié via un tube FIFO.
quitter         Quitte le programme.
MoncefSHELL >> 
```