Département
Informatique

RAPPORT DE MINI-PROJET SDA

Encadré par :

Mme Touhami Ouazzani Khadija
M Lazrek Mohamed

Réalisé par :

Touil Zouheir
Kassel Mohammed Issam
Achouch Mounia
Ait Aziz Amina

2020

Il apparaît opportun de commencer ce rapport par des remerciements à ceux qui ont eu la gentillesse de le rendre très profitable. Au terme de ce travail, Nous tenons à exprimer nos vifs remerciements à :

- Mme Touhami Ouazzani Khadija
- Mr Lazrek Mohamed

Nous vous remercions d'avoir partagé avec nous votre passion pour l'enseignement. Nous avons grandement apprécié votre soutien, votre implication et votre expérience tout au long de l'année.

Sommaire

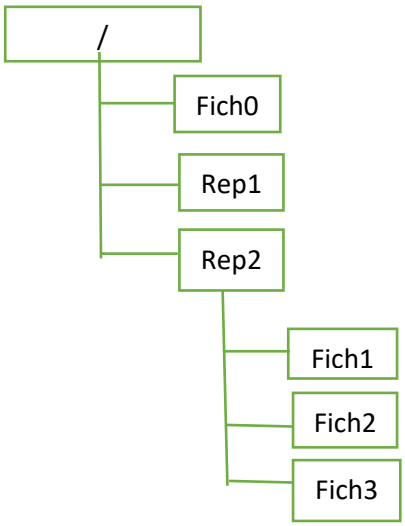
Description du Projet :

L'objectif du projet Structure de données avancées est de mettre en œuvre un système de fichiers hiérarchique simple avec stockage uniquement dans la mémoire principale. Un système de fichiers hiérarchique organise les ressources selon une structure arborescente et identifie de manière unique chaque ressource via le chemin qui la connecte à la racine. Les ressources contenues dans un système de fichiers hiérarchique peuvent être des fichiers ou des répertoires. Les deux ont un nom, représenté par une chaîne de caractères. Pour les premiers, il est seulement possible de les insérer comme feuilles de l'arbre, tandis que les seconds peuvent apparaître à la fois comme feuilles et comme nœuds intermédiaires. La racine de l'arbre est classiquement un répertoire, appelé répertoire racine. Seuls les nœuds de fichiers peuvent contenir des données, représentées comme une séquence d'octets, tandis que les répertoires n'ont pas de données associées. Tous les nœuds de l'arborescence peuvent contenir des métadonnées, mais aux besoins de ce projet, seuls les répertoires les contiennent. Les métadonnées du répertoire sont les noms de ses descendants directs.

Le programme qui implémente le système de fichiers recevra un journal des actions à effectuer à partir de l'entrée standard et imprimera le résultat de la même chose sur la sortie standard. Le programme doit être implémenté en C standard. Le fonctionnement du programme nécessite qu'il lise une ligne du journal des actions, et exécute l'action correspondante sur la représentation interne du système de fichiers qu'il gère, et écrive le résultat sur la sortie standard avant de passer à l'action suivante (l'exécution des actions est complètement séquentielle).

Travail réalisé :

Les chemins du système de fichiers sont représentés avec la syntaxe UNIX habituelle : un chemin est donc la séquence de noms de ressources qui du répertoire racine atteignent la ressource identifiée par le chemin. Les noms sont séparés par le caractère séparateur de chemin /). Par exemple, considérons le système de fichiers suivant :



Le chemin qui identifie la ressource de fichier 0 est / Fich0, celui qui identifie le fichier 3 est / Rep2 / Fich3.

La structure d'arbre n'aire s'applique.

Le programme reçoit l'une des commandes suivantes pour chaque ligne du fichier journal donné en entrée, où il indique un chemin générique et une chaîne alphanumérique avec un maximum de 255 caractères.

- Créer-f : la commande crée un fichier vide ou sans données associées dans le système de fichiers. Imprimer le résultat "fichier créé avec succès" si le fichier a été créé régulièrement, "Impossible de créer le fichier" si la création du fichier n'a pas réussi (par exemple, si vous

essayez de créer un fichier dans un répertoire inexistant ou si le nom du fichier existe déjà).

- Créer-rep : créez un répertoire vide dans le système de fichiers. Imprimez le résultat " répertoire créé avec succès " si le répertoire a été créé régulièrement, " Impossible de créer le répertoire" si la création n'a pas réussi.
- Lister : liste tous les éléments fichiers et répertoire du répertoire courant.
- Lister-D : liste tous répertoires du répertoire courant.
- Lister-F : liste tous fichiers du répertoire courant.
- Renommer <old name><new name> : modifie le nom d'un fichier ou d'un répertoire.
- Cd<path> : prend le chemin et passe d'un répertoire a un autre. Aussi on peur naviguer avec le Path et naviguer vers les parents.
- Rechercher <nom >: recherche toutes les ressources portant le nom spécifiés dans le système de fichiers sinon un erreur s'affiche.
- Supprimer : supprime un fichier ou une ressource et tous ses descendants, le cas échéant.
- Supprimer -f: supprimer un répertoire non vide , imprimer le résultat ("répertoire supprime avec succès ").
- chemin : affiche l'emplacement de la ressource dans le système de fichiers en affichant le chemin absolue vers le répertoire courant.
- Copier <nom><Path> : copie un fichier dans un autre répertoire avec le même nom.
- Help -- : affiche toutes les fonctions/commande du système.
- Quitter : met fin à l'exécution du gestionnaire de fichier

Code Source :

>> La structure d'un nœud d'arbre

```
typedef struct node node;
struct node{
    char name[40];
    char type;
    node* P; //pere
    node* FC; //les fils
    node* NS; // les freres
};
```

>>La structure de l'arbre d'un système de fichiers

```
struct filesystem{
    node* root;
    node* CD;
};
```

>>Initialiser l'arbre (création de la racine)

```
typedef struct filesystem fs;
fs* newFileSystem(){
    fs* f = (fs*)malloc(sizeof(fs));
    f -> root = (node*)malloc(sizeof(node));
    strcpy(f -> root -> name, "/");
    f -> root -> FC = NULL;
    f -> root -> NS = NULL;
    f -> root -> P = NULL;
    strcpy(f -> root -> name, "/");
    f -> CD = f -> root;
    return f;
}
```

>>chemin absolu récursive

```
char* recPWD(node* n, char* PWD, int first, int finder){
    char* temp;
    char temp2[21];
    if(n != NULL){
        temp = strdup(PWD);  //strdup returne un pointeur
        if(strcmp(n -> name, "/") != 0 && first > 0){
            strcpy(temp2, n -> name);
            strcat(temp2, "/");
            strcpy(PWD, temp2);
        }
        else{
            printf('\0');
            strcpy(PWD, n -> name);
        }
        strcat(PWD, temp);
        recPWD(n -> P, PWD, 1, finder);
    }
    free(temp);
    return PWD;
}


char* pwd(fs* f,int finder){
    char PWD[1000];
    memset(PWD, 0, 1000);
    if(f -> CD == f -> root){
        strcpy(PWD, "/");
        return PWD;
    }
    return recPWD(f -> CD, PWD, 0, finder);
}
```

>>comparer 2 noeuds (type / nom / taille)

```
int comparison(node* n1, node* n2){
    int size, x = 0;
    if(n1 -> type == 'D'){
        if(n2 -> type == 'F')
            return -1;
    }
    if(n1 -> type == 'F'){
        if(n2 -> type == 'D')
            return 1;
    }
    if(strcmp(n1 -> name, n2 -> name) == 0)
        return 0;
    if (strlen(n1 -> name) < strlen(n2 -> name))
        return -1;
    else if(strlen(n1 -> name) > strlen(n2 -> name))
        return 1;
    else
        size = strlen(n1 -> name);
    for(x = 0; x < size - 1; x++){
        char c1 = n1 -> name[x];
        char c2 = n2 -> name[x];
        if(c1 == c2)
            continue;
        if(c1 == '\0')
            return -1;
        if(c2 == '\0')
            return 1;
        if(c1 == '.')
            return -1;
        if(c2 == '.')
            return 1;
        if(c1 == '-')
            return -1;
        if(c2 == '-')
            return 1;
```

```
            return 1;
        if(c1 == '_')
            return -1;
        if(c2 == '_')
            return 1;
        if(c1 < c2)
            return -1;
        if(c1 > c2)
            return 1;
    }
    return 0;
}
```

>>fonction pour créer un répertoire

```
void mkdir(fs* f, char* path){
    char* token = strtok(path, "/");
    node* temp = f -> root -> FC;
    while(1){
        if(temp == NULL){ // si le systeme de fichiers est vide autre que root
            temp = (node*)malloc(sizeof(node));
            node* newDir = (node*)malloc(sizeof(node));
            strcpy(newDir -> name, token);
            newDir -> type = 'D';
            newDir -> FC = NULL;
            newDir -> NS = NULL;
            newDir -> P = f -> root;
            f -> root -> FC = newDir;
            recAddDir(f -> root -> FC, token);
                        printf("\033[1;32m");
                    printf("repertoire creer avec succes.\n");
                    printf("\033[0m");
            return;
        }

        if(strcmp(temp -> name, token) == 0){
            if(temp -> FC == NULL){ // correspondance trouvee, mais le repertoire trouve est vide
                token = strtok(NULL, "/");
                if(token == NULL){
                                        printf("\033[1;31m");
                    printf("impossible de creer le repertoire.\n");
                                        printf("\033[0m");
                    return;
                }
                node* newDir = (node*)malloc(sizeof(node));
                strcpy(newDir -> name, token);
                newDir -> type = 'D';
                newDir -> FC = NULL;
                newDir -> NS = NULL;
                newDir -> P = temp;
                temp -> FC = newDir;
                temp = temp -> FC;
```

```
            temp = temp -> FC;
            recAddDir(temp, token);
                        printf("\033[1;32m");
                printf("repertoire creer avec succes.\n");
                printf("\033[0m");

            return;
        }
        else{ // correspondance trouvee dans le repertoire courant, continuez vers l'interieur
            temp = temp -> FC;
            token = strtok(NULL, "/");
        }
    }
    else{ //correspondance introuvable, regardez l'entree suivante dans le repertoire courant
        if(temp -> NS == NULL){ //repertoire courant epuise, ajoutez le repertoire dans le repertoire courant
            node* newDir = (node*)malloc(sizeof(node));
            strcpy(newDir -> name, token);
            newDir -> type = 'D';
            newDir -> FC = NULL;
            newDir -> NS = NULL;
            newDir -> P = temp -> P;
             printf("\033[1;32m");
                printf("repertoire creer avec succes.\n");
                printf("\033[0m");
            ////// ajoutez//////
            node* temp2 = temp -> P -> FC;
            int added = 0;
            newDir -> NS = temp2; //ajouter au debut du repertoire courant
            temp -> P -> FC = newDir;
            node* prev = newDir -> NS;
            while(newDir -> NS != NULL){
                if(comparison(newDir, newDir -> NS) == 1){ //si la priorite est inferieure, descendez dans la liste
                    if(newDir == temp -> P -> FC){
                        temp -> P -> FC = temp2;
                    }
                    node* temp3 = newDir -> NS -> NS;
                    node* temp4 = newDir -> NS;
```

```
                    node* temp4 = newDir -> NS;
                    newDir -> NS -> NS = newDir;
                    newDir -> NS = temp3;
                    if(prev != temp4){
                        prev -> NS = temp4;
                        prev = prev -> NS;
                    }

                    added = 1;
                }
                else{

                    added = 1;
                    break;
                }
            }
            ////// ORDERING //////
            if(added == 0){
                temp -> NS = newDir;
                temp = temp -> NS;
            }
            recAddDir(temp, token);
            return;
        }
        else
            temp = temp -> NS;
    }
}
}
```

>>fonction de création des répertoires récursivement

```
void recAddDir(node* n, char* token){
    token = strtok(NULL, "/");
    if(token != NULL){
        node* newNode = (node*)malloc(sizeof(node));
        strcpy(newNode -> name, token);
        newNode -> type = 'D';
        newNode -> FC = NULL;
        newNode -> NS = NULL;
        newNode -> P = n;
        n -> FC = newNode;
        recAddDir(n -> FC, token);
    }
}
```

>>fonction pour créer un fichier

```
void touch(fs* f, char* path){
    char* token = strtok(path, "/");
    node* temp = f -> root -> FC;
    while(1){
        if(temp == NULL){ //si le systeme de fichiers est vide autre que root
            temp = (node*)malloc(sizeof(node));
            node* newFile = (node*)malloc(sizeof(node));  //pour creer le systeme de fichier
            strcpy(newFile -> name, token);
            newFile -> type = 'D';
            newFile -> FC = NULL;
            newFile -> NS = NULL;
            newFile -> P = f -> root;
            f -> root -> FC = newFile;
            token = strtok(NULL, "/"); ///
            recAddFile(f -> root -> FC, token);  //creation des fichier recursivement

            return;
        }
        if(strcmp(temp -> name, token) == 0){
            if(temp -> FC == NULL){ //correspondance trouvee, mais le repertoire trouve est vide
                token = strtok(NULL, "/");
                if(token == NULL){
                    printf("\033[1;31m");
                    printf("impossible de creer le fichier.\n");
                    printf("\033[0m");
                    return;
                }
                node* newFile = (node*)malloc(sizeof(node));
                strcpy(newFile -> name, token);
                newFile -> type = 'D';
                newFile -> FC = NULL;
                newFile -> NS = NULL;
                newFile -> P = temp;
                temp -> FC = newFile;
                temp = temp -> FC;
                token = strtok(NULL, "/"); ///
                recAddFile(temp, token);
```

```
                token = strtok(NULL, "/"); ///
                recAddFile(temp, token);
                 printf("\033[1;32m");
                    printf("fichier creer avec succes.\n");
                    printf("\033[0m");
                return;
            }
            else{ //correspondance trouvee dans le repertoire courant, continuez vers l'interieur
                temp = temp -> FC;
                token = strtok(NULL, "/");
            }
        }
        else{ //correspondance introuvable, regardez l'entree suivante dans le repertoire courant
            if(temp -> NS == NULL){ //repertoire courant epuise, ajoutez le repertoire dans le repertoire courant
                node* newFile = (node*)malloc(sizeof(node));
                strcpy(newFile -> name, token);
                token = strtok(NULL, "/");
                if(token == NULL)
                    newFile -> type = 'F';
                else
                    newFile -> type = 'D';
                newFile -> FC = NULL;
                newFile -> NS = NULL;
                newFile -> P = temp -> P;
                printf("\033[1;32m");
                    printf("fichier creer avec succes.\n");
                    printf("\033[0m");
                ////// ORDERING //////
                node* temp2 = temp -> P -> FC;
                int added = 0;
                newFile -> NS = temp2; // ajouter au debut du repertoire courant
                temp -> P -> FC = newFile;
                node* prev = newFile -> NS;
                while(newFile -> NS != NULL){
                    if(comparison(newFile, newFile -> NS) == 1){ //
//si la priorite est inferieure, descendez dans la liste
                        if(newFile == temp -> P -> FC){
```

```
                            temp -> P -> FC = temp2;
                        }
                        node* temp3 = newFile -> NS -> NS;
                        node* temp4 = newFile -> NS;
                        newFile -> NS -> NS = newFile;
                        newFile -> NS = temp3;
                        if(prev != temp4){
                            prev -> NS = temp4;
                            prev = prev -> NS;
                        }

                        added = 1;
                    }
                    else{

                        added = 1;
                        break;
                    }
                }
                ////// ORDERING //////
                if(added == 0){
                    temp -> NS = newFile;
                    temp = temp -> NS;
                }
                recAddFile(newFile, token);
                return;
            }
            else
                temp = temp -> NS;
        }
    }
}
void copier(fs* f, char* path){
    char* token = strtok(path, "/");
    node* temp = f -> root -> FC;
    while(1){
        if(temp == NULL){ //si le systeme de fichiers est vide autre que root
```

```
        if(temp == NULL){ //si le systeme de fichiers est vide autre que root
            temp = (node*)malloc(sizeof(node));
            node* newFile = (node*)malloc(sizeof(node));  //pour creer le systeme de fichier
            strcpy(newFile -> name, token);
            newFile -> type = 'D';
            newFile -> FC = NULL;
            newFile -> NS = NULL;
            newFile -> P = f -> root;
            f -> root -> FC = newFile;
            token = strtok(NULL, "/"); ///
            recAddFile(f -> root -> FC, token);  //creation dans fichier recursivement

            return;
        }
        if(strcmp(temp -> name, token) == 0){
            if(temp -> FC == NULL){ //correspondance trouvee, mais le repertoire trouve est vide
                token = strtok(NULL, "/");
                if(token == NULL){
                    printf("\033[1;31m");
                    printf("impossible de copier le fichier.\n");
                    printf("\033[0m");
                    return;
                }
                node* newFile = (node*)malloc(sizeof(node));
                strcpy(newFile -> name, token);
                newFile -> type = 'D';
                newFile -> FC = NULL;
                newFile -> NS = NULL;
                newFile -> P = temp;
                temp -> FC = newFile;
                temp = temp -> FC;
                token = strtok(NULL, "/"); ///
                recAddFile(temp, token);
                 printf("\033[1;32m");
                    printf("fichier a ete copie avec succes.\n");
                    printf("\033[0m");
                return;
```

```
            }
            else{ //correspondance trouvee dans le repertoire courant, continuez vers l'interieur
                temp = temp -> FC;
                token = strtok(NULL, "/");
            }
        }
        else{ //correspondance introuvable, regardez l'entree suivante dans le repertoire courant
            if(temp -> NS == NULL){ //repertoire courant epuise, ajoutez le repertoire dans le repertoire courant
                node* newFile = (node*)malloc(sizeof(node));
                strcpy(newFile -> name, token);
                token = strtok(NULL, "/");
                if(token == NULL)
                    newFile -> type = 'F';
                else
                    newFile -> type = 'D';
                newFile -> FC = NULL;
                newFile -> NS = NULL;
                newFile -> P = temp -> P;
                printf("\033[1;32m");
                    printf("fichier a ete copie avec succes.\n");
                    printf("\033[0m");
                ////// ORDERING //////
                node* temp2 = temp -> P -> FC;
                int added = 0;
                newFile -> NS = temp2; // ajoutez au debut du repertoire courant
                temp -> P -> FC = newFile;
                node* prev = newFile -> NS;
                while(newFile -> NS != NULL){
                    if(comparison(newFile, newFile -> NS) == 1){ //
//si la priorite est inferieure, descendez dans la liste
                        if(newFile == temp -> P -> FC){
                            temp -> P -> FC = temp2;
                        }
                        node* temp3 = newFile -> NS -> NS;
                        node* temp4 = newFile -> NS;
                        newFile -> NS -> NS = newFile;
                        newFile -> NS = temp3;
```

```
                    newFile -> NS = temp3;
                    if(prev != temp4){
                        prev -> NS = temp4;
                        prev = prev -> NS;
                    }

                    added = 1;
                }
                else{

                    added = 1;
                    break;
                }
            }
            ////// ORDERING //////
            if(added == 0){
                temp -> NS = newFile;
                temp = temp -> NS;
            }
            recAddFile(newFile, token);
            return;
        }
        else
            temp = temp -> NS;
    }
}
```

>>la fonction de création des fichiers récursivement

```
void recAddFile(node* n, char* token){
    if(token == NULL){
        n -> type = 'F';
    }
    if(token != NULL){
        node* newNode = (node*)malloc(sizeof(node));
        strcpy(newNode -> name, token);
        newNode -> type = 'D';
        newNode -> FC = NULL;
        newNode -> NS = NULL;
        newNode -> P = n;
        n -> FC = newNode;
        token = strtok(NULL, "/");
        recAddFile(n -> FC, token);
    }
}
```

>>la fonction pour ordonner le system de fichier

```
void order(fs* f){    //entree: un systeme de fichier
    recOrder(f -> root -> FC);
}
```

>>la fonction pour ordonner récursivement le system de fichier

```
void recOrder(node* n){
    if(n -> NS != NULL){
        printf("%s", n -> name);
        if(comparison(n, n -> NS) == 1){
            node* temp = n;
            n = n -> NS;
            n -> NS = temp;
            recOrder(n);
        }
    }
    if(n -> NS != NULL){
        recOrder(n -> NS);
    }
}
```

>>la fonction rechercher d'un fichier ou répertoire

```
void find(fs* f, char* name){
    printf("A la recherche de '%s':\n ", name);
    printf("\e[0;101m");
     printf(" \t(s'il exist vous aurez tout les chemins absolus de cet element :>; sinon vous aurez rien :<)\n");
     printf("\033[0m");
    recFind(f,f -> root -> FC, name);
}
```

>>la fonction rechercher recursivement d'un fichier ou repertoire

```
void recFind(fs* f, node* n, char* name){
    findHelp(f, n, name);
    if(n -> FC != NULL){
        recFind(f, n -> FC, name);
    }
    if(n -> NS != NULL){
        recFind(f, n -> NS, name);
    }

}
```

>>fonction pour l'exécution de la recherche récursive

```
void findHelp(fs* f, node* n, char* name){
    int i = 0;
    int j = 0;
    int c = 0;
    char PWD[1000];
    memset(PWD, 0, 1000);
    int complete = strlen(name);
    for(i = 0; i < strlen(n -> name); i++){
        if(n -> name[i] == name[j]){
            c++;
            j++;
            if(c == complete){
                node* temp = f -> CD;
                f -> CD = n;
                strcpy(PWD, pwd(f,1));
                printf("%c ", n -> type);
                printf("%s\n", PWD);
                f -> CD = temp;

                break;
                    }
        }
    }
}
void renommer(fs* f, char* path){
    char* token = strtok(path, "/");
    node* temp = f -> root -> FC;
    while(1){
        if(temp == NULL){ //si le systeme de fichiers est vide autre que root
            temp = (node*)malloc(sizeof(node));
            node* newFile = (node*)malloc(sizeof(node));  //pour creer le systeme de fichier
            strcpy(newFile -> name, token);
            newFile -> type = 'D';
            newFile -> FC = NULL;
            newFile -> NS = NULL;
            newFile -> P = f -> root;
```

```
            newFile -> P = f -> root;
            f -> root -> FC = newFile;
            token = strtok(NULL, "/"); ///
            recAddFile(f -> root -> FC, token);  //creation des fichier recursivement

            return;
        }
        if(strcmp(temp -> name, token) == 0){
            if(temp -> FC == NULL){ //correspondance trouvee, mais le repertoire trouve est vide
                token = strtok(NULL, "/");
                if(token == NULL){
                    printf("\033[1;31m");
                    printf("impossible de renommer le fichier.\n");
                    printf("\033[0m");
                    return;
                }
                node* newFile = (node*)malloc(sizeof(node));
                strcpy(newFile -> name, token);
                newFile -> type = 'D';
                newFile -> FC = NULL;
                newFile -> NS = NULL;
                newFile -> P = temp;
                temp -> FC = newFile;
                temp = temp -> FC;
                token = strtok(NULL, "/"); ///
                recAddFile(temp, token);
                 printf("\033[1;32m");
                    printf("fichier renommer avec succes.\n");
                    printf("\033[0m");
                return;
            }
            else{ //correspondance trouvee dans le repertoire courant, continuez vers l'interieur
                temp = temp -> FC;
                token = strtok(NULL, "/");
            }
        }
        else{ //correspondance introuvable, regardez l'entree suivante dans le repertoire courant
```

```
        else{ //correspondance introuvable, regardez l'entree suivante dans le repertoire courant
            if(temp -> NS == NULL){ //repertoire courant vide, ajoutez le repertoire dans le repertoire courant
                node* newFile = (node*)malloc(sizeof(node));
                strcpy(newFile -> name, token);
                token = strtok(NULL, "/");
                if(token == NULL)
                    newFile -> type = 'F';
                else
                    newFile -> type = 'D';
                newFile -> FC = NULL;
                newFile -> NS = NULL;
                newFile -> P = temp -> P;
                printf("\033[1;32m");
                    printf("fichier renommer avec succes.\n");
                    printf("\033[0m");
                ////// ORDERING //////
                node* temp2 = temp -> P -> FC;
                int added = 0;
                newFile -> NS = temp2; // ajouter au debut du repertoire courant
                temp -> P -> FC = newFile;
                node* prev = newFile -> NS;
                while(newFile -> NS != NULL){
                    if(comparison(newFile, newFile -> NS) == 1){ //
//si la priorite est inferieure, descendez dans la liste
                        if(newFile == temp -> P -> FC){
                            temp -> P -> FC = temp2;
                        }
                        node* temp3 = newFile -> NS -> NS;
                        node* temp4 = newFile -> NS;
                        newFile -> NS -> NS = newFile;
                        newFile -> NS = temp3;
                        if(prev != temp4){
                            prev -> NS = temp4;
                            prev = prev -> NS;
                        }

                        added = 1;
```

```
                    added = 1;
                }
                else{

                    added = 1;
                    break;
                }
            }
            ////// ORDERING //////
            if(added == 0){
                temp -> NS = newFile;
                temp = temp -> NS;
            }
            recAddFile(newFile, token);
            return;
        }
        else
            temp = temp -> NS;
    }
}
```

>>La fonction renommer

```
void renommerrec(fs* f, char* path, int force){
    char* token = strtok(path, "/");
    node* temp = f -> root -> FC;
    while(1){
        if(temp == NULL){
            return;
        }
        node* temp2 = f -> root -> FC;
        node* temp3 = f -> root -> FC;
        if(strcmp(temp -> name, token) == 0){
            if(temp -> FC == NULL){
                token = strtok(NULL, "/");
                if(token == NULL){
                    node* parent = temp -> P;
                    temp2=temp->NS;
                    parent -> FC = NULL;
                    parent->FC=temp2;
                    free(temp);

                    return;
                }
                else{
                    return;
                }
            }
            else{
                token = strtok(NULL, "/");
                if(token == NULL){
                    if(force == 1){
                        recRemove(temp);
                        return;
                    }
                    else{
                        return;
                    }
                }
                temp = temp -> FC;
```

```
            }
        }
        else{
            if(temp -> NS == NULL){

                return;
            }
            else{
                temp = temp -> NS;
            }
        }
    }
}
```

>>supprimer un fichier ou un répertoire

```
void rm(fs* f, char* path, int force){
    char* token = strtok(path, "/");
    node* temp = f -> root -> FC;
    while(1){
        if(temp == NULL){
                        printf("\033[1;31m");
            printf("impossible de supprimer le fichier ou le repertoire.\n");
                        printf("\033[0m");
            return;
        }
        node* temp2 = f -> root -> FC;
        node* temp3 = f -> root -> FC;
        if(strcmp(temp -> name, token) == 0){
            if(temp -> FC == NULL){
                token = strtok(NULL, "/");
                if(token == NULL){
                    node* parent = temp -> P;
                    temp2=temp->NS;
                    parent -> FC = NULL;
                    parent->FC=temp2;
                    free(temp);

                    return;
                }
                else{   printf("\033[1;31m");
                    printf("impossible de supprimer le fichier ou le repertoire.\n");
                        printf("\033[0m");
                    return;
                }
            }
            else{
                token = strtok(NULL, "/");
                if(token == NULL){
                    if(force == 1){
                        recRemove(temp);
                        return;
                    }
```

```
                }
                else{   printf("\033[1;31m");
                    printf("impossible de supprimer le repertoire '%s' il n est pas vide.\n", path);
                        printf("\033[0m");
                    return;
                }
            }
            temp = temp -> FC;
        }
    }
    else{
        if(temp -> NS == NULL){
                        printf("\033[1;31m");
                    printf("impossible de supprimer le fichier ou le repertoire.\n");
                        printf("\033[0m");
            return;
        }
        else{
            temp = temp -> NS;
        }
    }
}
}
```

>>la fonction pour supprimer un élément récursivement

```
void recRemove(node* n){
    if(n -> FC != NULL){
        recRemove(n -> FC);
    }
    if(n -> NS != NULL){
        recRemove(n -> NS);
    }
    node* parent = n -> P;
    parent -> FC = NULL;
    free(n);

}
```

>>naviguer entre les répertoires

```
void cd(fs* f, char* path){
    if(strcmp(path, "/") == 0){
        f -> CD = f -> root;
        return;
    }
    char* token = strtok(path, "/");
    node* temp = f -> root -> FC;

    while(1){
        if(token == NULL || (strcmp(token, ".") != 0 && strcmp(token, "..") != 0)){ //retourner a la methods cd normale
            break;
        }
        if(strcmp(token, ".") == 0){
            temp = f -> CD;
            token = strtok(NULL, "/");
            if(token == NULL){
                f -> CD = temp;
                return;
            }
        }
        else if(strcmp(token, "..") == 0){
            if(f -> CD == f -> root)
                temp = f -> root;
            else{
                temp = f -> CD -> P;
                f -> CD = f -> CD -> P;
            }
            token = strtok(NULL, "/");
            if(token == NULL){
                f -> CD = temp;
                return;
            }
        }
    }
```

```
    while(1){
        if(temp == NULL){
                        printf("\033[1;31m");
            printf("impossible de changer le repertoire .\n");
                        printf("\033[0m");
            return;
        }
        if(strcmp(temp -> name, token) == 0){
            if(temp -> FC == NULL){
                token = strtok(NULL, "/");
                if(token == NULL){
                    if(temp -> type == 'D'){
                        f -> CD = temp;
                        return;
                    }
                    else{   printf("\033[1;31m");
                        printf("impossible de changer le repertoire .\n");
                            printf("\033[0m");
                        return;
                    }
                }
                else{   printf("\033[1;31m");
                    printf("impossible de changer le repertoire .\n");
                        printf("\033[0m");
                    return;
                }
            }
            else{
                token = strtok(NULL, "/");
                if(token == NULL){
                    if(temp -> type == 'D'){
                        f -> CD = temp;
                        return;
                    }
                    else{   printf("\033[1;31m");
                        printf("impossible de changer le repertoire .\n");
                            printf("\033[0m");
```

```
                    }
                    temp = temp -> FC;
                }
            }
            else{
                if(temp -> NS == NULL){
                                printf("\033[1;31m");
                    printf("impossible de changer le repertoire .\n");
                                printf("\033[0m");
                    return;
                }
                else{
                    temp = temp -> NS;
                }
            }
        }
    }
```

>>lister le contenu d'un répertoire

```
void ls(fs* f, char* path){
    if(strcmp(path, "") == 0){
        node* temp = f -> CD -> FC;
        char* PWD = pwd(f,0);
        printf("Listing For '%s':\n", PWD);
        while(temp != NULL){
            if(temp -> type == 'D'){
                printf("\033[1;34m");
                printf("D ");
                printf("%s\n", temp -> name);
                temp = temp -> NS;
                printf("\033[0m");
            }
            else
                {
                    printf("\033[1;36m");
                printf("F ");
                printf("%s\n", temp -> name);
                temp = temp -> NS;
                 printf("\033[0m");
                }
        }
    }
    else{
        char* token = strtok(path, "/");
        node* temp = f -> root -> FC;
        while(1){
            if(temp == NULL){
                                printf("\033[1;31m");
                printf("impossible de lister les elements du repertoire.\n");
                                printf("\033[0m");
                return;
            }
            if(strcmp(token, ".") == 0){
                temp = f -> CD;
                token = strtok(NULL, "/");
                if(token == NULL)
```

```
                if(token == NULL)
                    break;
            }
            if(strcmp(token, "..") == 0){
                if(f -> CD == f -> root){
                    temp = f -> root;
                }
                else{
                    temp = f -> CD -> P;
                    f -> CD = f -> CD -> P;
                }
                token = strtok(NULL, "/");
                if(token == NULL)
                    break;
            }
            if(strcmp(temp -> name, token) == 0){
                if(temp -> FC == NULL){ //correspondance trouvŽe, mais le rŽpertoire trouvŽ est vide
                    token = strtok(NULL, "/");
                    if(token == NULL)
                        break;
                    else{   printf("\033[1;31m");
                        printf("impossible de lister les elements du repertoire.\n");
                            printf("\033[0m");
                        return;
                    }
                }
                else{ //
//correspondance trouvee dans le repertoire courant, continuez a l'interieur
                    token = strtok(NULL, "/");
                    if(token == NULL){
                        break;
                    }
                    temp = temp -> FC;
                }
            }
            else{ //correspondance introuvable, regardez l'entree suivante dans le repertoire courant
                if(temp -> NS == NULL){ //repertoire courant epuise, ajoutez le repertoire dans le repertoire courant
```

```
                if(temp -> NS == NULL){ //repertoire courant epuise, ajoutez le repertoire dans le repertoire courant
                        printf("\033[1;31m");
                    printf("impossible de lister les elements du repertoire.\n");
                        printf("\033[0m");
                    return;
                }
                else
                    temp = temp -> NS;
            }
        }
        if(temp -> type == 'F'){
                        printf("\033[1;31m");
            printf("impossible de lister les elements. '%s' est un fichier.\n", path);
                        printf("\033[0m");
            return;
        }
        else{
            printf("lister les elements pour : '/%s':\n", path);
            temp = temp -> FC;
            while(temp != NULL){
                if(temp -> type == 'D')
                {
                    printf("\033[1;34m");
                printf("D ");
                printf("%s\n", temp -> name);
                temp = temp -> NS;
                printf("\033[0m");
                }
                else
                    {
                        printf("\033[1;36m");
                printf("F ");
                printf("%s\n", temp -> name);
                temp = temp -> NS;
                 printf("\033[0m");
                    }
```

```
        }
    }
}
void ls1(fs* f, char* path){
    if(strcmp(path, "") == 0){
        node* temp = f -> CD -> FC;
        char* PWD = pwd(f,0);
        printf("Listing For '%s':\n", PWD);
        while(temp != NULL){
            if(temp -> type == 'D'){
                printf("\033[1;34m");
                printf("D ");
                printf("%s\n", temp -> name);
                printf("\033[0m");
                }
                temp = temp -> NS;

        }
    }

}
void ls2(fs* f, char* path){
    if(strcmp(path, "") == 0){
        node* temp = f -> CD -> FC;
        char* PWD = pwd(f,0);
        printf("Listing For '%s':\n", PWD);
        while(temp != NULL){
            if(temp -> type == 'F'){
            printf("\033[1;36m");
                printf("F ");
                printf("%s\n", temp -> name);
                 printf("\033[0m");
                }
                temp = temp -> NS;

        }
    }
```

>>la fonction HELP

```
void HELP(char* str){
    if(strcmp(str, "creer-f") == 0)
    {

        printf("\t=============================================================================================================\n");
        printf("\t| creer-f <nom>:  cree un fichier vide, sans aucune donnee associee.                                        |\n");
        printf("\t=============================================================================================================\n");
    }
    if(strcmp(str, "creer-rep") == 0){
        printf("\t=============================================================================================================\n");
        printf("\t| creer-rep <nom>:   cree un repertoire vide, sans aucun fils.                                              |\n");
        printf("\t=============================================================================================================\n");
         printf("\t=============================================================================================================\n");
        printf("\t|NOTE:cette commande  affichera une erreur si la ressource specifiee n'existe pas.                         |\n");
        printf("\t=============================================================================================================\n");
    }
    if(strcmp(str, "rechercher") == 0){
        printf("\t=============================================================================================================\n");
        printf("\t| rechercher <nom>:    recherche toutes les ressources portant le nom specifie dans le systeme de fichiers. |\n");
        printf("\t=============================================================================================================\n");
        printf("\t=============================================================================================================\n");
        printf("\t|NOTE: <rechercher> elle donne Le chemin de toutes les ressources rencontrees .                             \n");
        printf("\t=============================================================================================================\n");
    }
    if(strcmp(str, "chemin") == 0){
        printf("\t=============================================================================================================\n");
        printf("\t| chemin:    affiche le chemin vers le repertoire courant.                                                  |\n");
        printf("\t=============================================================================================================\n");
    }
    if(strcmp(str, "lister") == 0){
        printf("\t=============================================================================================================\n");
        printf("\t| lister:    liste tous les elements du repertoire courant.avec d'autres parametres  -D/-F/..               |\n");
        printf("\t=============================================================================================================\n");
    }
    if(strcmp(str,"--")==0)
        {
printf("La structure du systeme de fichiers peut distinguer 11 commandes differentes:\n");
```



```
        {
printf("La structure du systeme de fichiers peut distinguer 11 commandes differentes:\n");
printf("\n");
printf("\t=============================================================================================================\n");
printf("\t| creer-f <nom>:  cree un fichier vide, sans aucune donnee associee.                                        |\n");
printf("\t=============================================================================================================\n");
printf("\t| creer-rep <nom>:   cree un repertoire vide, sans aucun fils.                                              |\n");
printf("\t=============================================================================================================\n");
printf("\t| cd <nom>:   prend le chemin et passe d'un repertoire a un autre.                                          |\n");
printf("\t=============================================================================================================\n");
printf("\t| copier <nom> <path>:    copie un fichier dans le path indique                                             |\n");
printf("\t=============================================================================================================\n");
printf("\t| supprimer <nome>:supprime un fichier ou bien repertoire vide !!                                           |\n");
printf("\t=============================================================================================================\n");
printf("\t| supprimer -f <nom>:    supprime recursivement un repertoire et tous ses descendants.                      |\n");
printf("\t=============================================================================================================\n");
printf("\t|NOTE:cette commande  affichera une erreur si la ressource specifiee n'existe pas.                          |\n");
printf("\t=============================================================================================================\n");
printf("\t| rechercher <nom>:    recherche toutes les element portant le nom specifie dans le systeme de fichiers.    |\n");
printf("\t=============================================================================================================\n");
printf("\t| renommer <oldname><newname>:     pour renommer un fichier et le donner le newname                         |\n");
printf("\t=============================================================================================================\n");
printf("\t| lister:    liste tous les elements du repertoire courant.Avec d'autre parametre -D/-F/..                  |\n");
printf("\t=============================================================================================================\n");
printf("\t| chemin:    affiche le chemin absolu vers le repertoire courant.                                           |\n");
printf("\t=============================================================================================================\n");
printf("\t| help --: si vous avez un probleme tapez help -- !!                                                        |\n");
printf("\t=============================================================================================================\n");
printf("\t|NOTE: Si vous tapez help <nome du commande> vous aurez les information necessaire !!                       |\n");
printf("\t=============================================================================================================\n");
printf("\t| quitter:    termine l'execution du programme de gestion du systeme de fichiers.                           |\n");
printf("\t=============================================================================================================\n");
                }
}
```

>>la fonction PRINCIPALE main

```c
int main(){
    char choix;
    int n=0;
    //la gestion de l'interface

do{     system ("cls");
        system ("Color 79 ");

        printf("\t\t\t\t\t@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n");
        printf("\t\t\t\t\t@@          ");
        printf("\033[1;31m");
        printf("mini-projet SDA");
        printf("\033[1;34m");
        printf("            @@\n");
        printf("\t\t\t\t\t||                                    ||\n");
        printf("\t\t\t\t\t@@  Encadre par:                      @@\n");
        printf("\t\t\t\t\t||     ");
        printf("\033[1;30m");
        printf("* Touhami Ouazzani Khadija");
        printf("\033[1;34m");
        printf("     ||\n");
        printf("\t\t\t\t\t||     ");
        printf("\033[1;30m");
        printf("* Lazrek mohamed          ");
        printf("\033[1;34m");
        printf("     ||\n");

        printf("\t\t\t\t\t||                                    ||\n");
        printf("\t\t\t\t\t@@   Realise par :                    @@\n");
        printf("\t\t\t\t\t||     ");
        printf("\033[1;30m");
        printf("* TOUIL ZOUHEIR           ");
        printf("\033[1;34m");
        printf("     ||\n");        printf("\t\t\t\t\t||     ");
        printf("\033[1;30m");
        printf("* KASSEL MAHEMMED ISSAM   ");
        printf("\033[1;34m");
```

```
        printf("\033[1;34m");
        printf("     ||\n");         printf("\t\t\t\t\t||     ");
        printf("\033[1;30m");
        printf("* AIT AZIZ AMINA          ");
        printf("\033[1;34m");
        printf("     ||\n");         printf("\t\t\t\t\t@@     ");
        printf("\033[1;30m");
        printf("* ACHOUCH MOUNIA          ");
        printf("\033[1;34m");
        printf("     @@\n");
        printf("\t\t\t\t\t||                                        ||\n");
        printf("\t\t\t\t\t@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@\n");
        printf("\033[1;31m");
        printf("\t\t\t\t\t        System de gestion de fichier     \n");
        printf("\033[1;34m");
        printf("\t\t\t\t\t_________________________________________\n");
        printf("\t\t\t\t\t| |                                     |\n");
        printf("\t\t\t\t\t|1|      Ouvrir le gestionnaire        |\n");
        printf("\t\t\t\t\t| |                                     |\n");
        printf("\t\t\t\t\t=======================================\n");
        printf("\t\t\t\t\t| |                                     |\n");
        printf("\t\t\t\t\t|2|     Les indications d'utilisation  |\n");
        printf("\t\t\t\t\t| |                                     |\n");
        printf("\t\t\t\t\t=======================================\n");
        printf("\t\t\t\t\t| |                                     |\n");
        printf("\t\t\t\t\t|3|      Quitter l'application         |\n");
        printf("\t\t\t\t\t|_|_____________________________________|\n");
        printf("\n\r Tapez svp votre choix ?  (1/2/3) \n");
        choix=toupper(getch());
        switch(choix){
            case '1' :system ("cls");
                      system ("color 7");
                         printf("\t\t\t\t\t_________________________________________\n");
                         printf("\t\t\t\t\t||                                     ||\n");
                         printf("\t\t\t\t\t||     System de gestion de fichier    ||\n");
                         printf("\t\t\t\t\t||                                     ||\n");
                         printf("\t\t\t\t\t=======================================\n");
```

```
fs* f = newFileSystem();
printf("\033[5;33m");
printf("\t\t\t\t\tBienvenu dans le systeme de fichiers :) !! \n" );
printf("\033[0m");

char command[20];
char path[1000];
char nom[1000];
char newname[40];
char name[40];
while(scanf("%s", command) > 0){
            if((strcmp(command, "renommer") != 0) && (strcmp(command, "copier") != 0)&& (strcmp(command, "creer-rep") != 0)
               && (strcmp(command, "creer-f") != 0) && (strcmp(command, "supprimer") != 0)
               && (strcmp(command, "supprimer -f") != 0) && (strcmp(command, "cd") != 0) && (strcmp(command, "chemin") != 0)
               && (strcmp(command, "lister") != 0 )&& (strcmp(command, "lister-D") != 0 )&& (strcmp(command, "lister-F") != 0 )
               && (strcmp(command, "rechercher") != 0) && (strcmp(command, "quitter") != 0) && (strcmp(command, "help") != 0 )){
        printf("\033[1;31m");
        printf("La Command %s est introuvable :( pour plus d'information tapez help -- !!\n",command);
        printf("\033[0m");
    }
    if(strcmp(command, "creer-rep") == 0){
    if(n==0){scanf("%s",path );mkdir(f, path);cd(f, path);} //la creation de l'arbre pour la 1 er fois
            else{
        scanf("%s",nom );
        char* PWD = pwd(f,0);
        int i;
        for (i = 0; PWD[i]!='\0'; i++);
        PWD[i]='/';
        i++;

        for (int j = 0; nom[j]!='\0'; j++, i++)
            {
                PWD[i] = nom[j];
            }
            PWD[i] = '\0';
        mkdir(f, PWD);
```

```
            mkdir(f, PWD);
                }
        }
         if(strcmp(command, "renommer") ==0){


            scanf("%s", nom);
            scanf("%s",newname );
            char* PWD = pwd(f,0);
            int i;
            for (i = 0; PWD[i]!='\0'; i++);
            PWD[i]='/';
            i++;

            for (int j = 0; nom[j]!='\0'; j++, i++)
                {
                    PWD[i] = nom[j];
                }
                PWD[i] = '\0';
                renommerrec(f, PWD, 0);


            char* PWDy = pwd(f,0);
            int k;
            for (k = 0; PWDy[k]!='\0'; k++);
            PWD[k]='/';
            k++;

            for (int j = 0; newname[j]!='\0'; j++, k++)
                {
                    PWDy[k] = newname[j];
                }
                PWDy[k] = '\0';
            renommer(f, PWDy);

                }
```

```
        if(strcmp(command, "creer-f") == 0){
            scanf("%s",nom );
            char* PWD = pwd(f,0);
            int i;
            for (i = 0; PWD[i]!='\0'; i++);
            PWD[i]='/';
            i++;

            for (int j = 0; nom[j]!='\0'; j++, i++)
            {
                PWD[i] = nom[j];
            }
            PWD[i] = '\0';
            touch(f, PWD);
        }

        if(strcmp(command, "supprimer") == 0){
            scanf("%s", nom);
            if(strcmp(nom, "-f") == 0){
            scanf("%s", nom);
            char* PWD = pwd(f,0);
            int i;
            for (i = 0; PWD[i]!='\0'; i++);
            PWD[i]='/';
            i++;

            for (int j = 0; nom[j]!='\0'; j++, i++)
            {
                PWD[i] = nom[j];
            }
            PWD[i] = '\0';
            rm(f, PWD, 1);
            }
            else{
            char* PWD = pwd(f,0);
            int i;
            for (i = 0; PWD[i]!='\0'; i++);
```

```
            PWD[i]='/';
            i++;

            for (int j = 0; nom[j]!='\0'; j++, i++)
                {
                    PWD[i] = nom[j];
                }
                PWD[i] = '\0';
                rm(f, PWD, 0);
            }
        }
        if(strcmp(command, "supprimer -f") == 0){
            scanf("%s", nom);
            char* PWD = pwd(f,0);
            int i;
            for (i = 0; PWD[i]!='\0'; i++);
            PWD[i]='/';
            i++;

            for (int j = 0; nom[j]!='\0'; j++, i++)
                {
                    PWD[i] = nom[j];
                }
                PWD[i] = '\0';
                rm(f, PWD, 1);
        }
        if(strcmp(command, "rechercher") == 0){
            scanf("%s", name);
            find(f, name);
        }
        if(strcmp(command, "cd") == 0){
            scanf("%s", nom);
                if(strcmp(nom, "..") != 0)
                {
                char* PWD = pwd(f,0);
                int i;
                for (i = 0; PWD[i]!='\0'; i++);
```

```
                PWD[i]='/';
                i++;

                for (int j = 0; nom[j]!='\0'; j++, i++)
                {
                    PWD[i] = nom[j];
                }
                PWD[i] = '\0';
               cd(f, PWD);
                }
                else
                    cd(f,nom);
        }

        if(strcmp(command, "quitter") == 0){
            system ("cls");
                        system ("Color 79");
                       printf("\t\t\t\t\t_____________________________________________\n");
                       printf("\t\t\t\t\t||                                         ||\n");
                       printf("\t\t\t\t\t||    Fin du Programme Gestionnair         ||\n");
                       printf("\t\t\t\t\t||                                         ||\n");
                       printf("\t\t\t\t\t=============================================\n");
                exit(0);
        }
        if(strcmp(command, "chemin") == 0){
            char* PWD = pwd(f,0);
            printf("%s\n", PWD);
        }
        if(strcmp(command, "lister-D") == 0){
            memset(path, 0, 1000);
            gets(path);
            if(path[0] == '0'){
                strcpy(path, "");
            }
            else{
                char* p = path;
                p += 1;
```

```
                strcpy(path, p);
            }
            lsl(f, path);
        }
        else
            if(strcmp(command, "lister-F") == 0){
            memset(path, 0, 1000);
            gets(path);
            if(path[0] == '0'){
                strcpy(path, "");
            }
            else{
                char* p = path;
                p += 1;
                strcpy(path, p);
            }
            ls2(f, path);
            }
        if(strcmp(command, "lister") == 0){
            memset(path, 0, 1000);
            gets(path);
            if(path[0] == '0'){
                strcpy(path, "");
            }
            else{
                char* p = path;
                p += 1;
                strcpy(path, p);
            }
            ls(f, path);
            }
            if(strcmp(command, "copier") == 0)
            {
                scanf("%s",nom );
                scanf("%s",path);
            int i;
            for (i = 0; path[i]!='\0'; i++);
```

```
            path[i]='/';
            i++;

            for (int j = 0; nom[j]!='\0'; j++, i++)
                {
                    path[i] = nom[j];
                }
                path[i] = '\0';
            copier(f, path);

            }



        if (strcmp(command, "help") == 0){
                        scanf("%s", path);
                        HELP(path);
                }
        if(n=1)
            {
                char* PWD = pwd(f,0);
            printf("%s  >>\t", PWD);
            }
            n=1;

    }
    case '2' : system ("cls");
                    system ("Color 79");
                       printf("\t\t\t\t\t_____________________________________________\n");
                       printf("\t\t\t\t\t||                                           ||\n");
                       printf("\t\t\t\t\t||    les indications d'utilisation          ||\n");
                       printf("\t\t\t\t\t||                                           ||\n");
                       printf("\t\t\t\t\t=============================================\n");
                       printf("\n");
printf("La structure du systeme de fichiers peut distinguer les commandes suivantes:\n");
printf("\n");
```

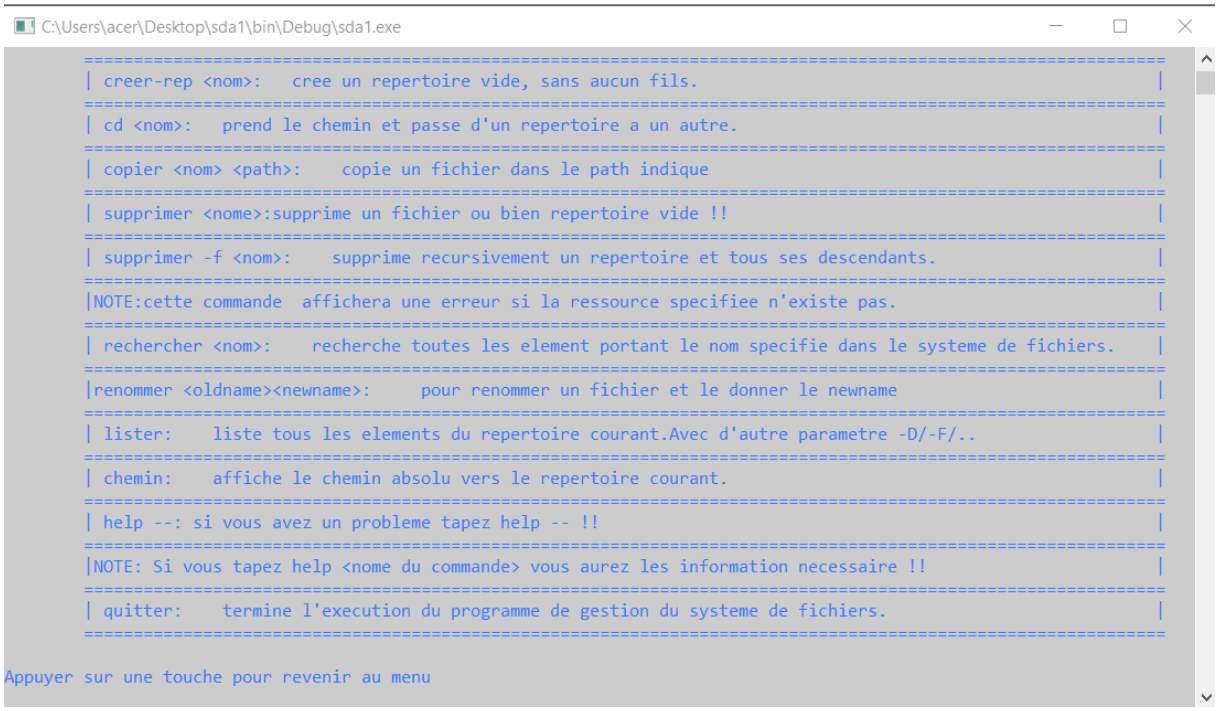
```
printf("La structure du systeme de fichiers peut distinguer les commandes suivantes:\n");
printf("\n");
printf("\t=============================================================================================================\n");
printf("\t| creer-f <nom>:  cree un fichier vide, sans aucune donnee associee.                                        |\n");
printf("\t=============================================================================================================\n");
printf("\t| creer-rep <nom>:   cree un repertoire vide, sans aucun fils.                                              |\n");
printf("\t=============================================================================================================\n");
printf("\t| cd <nom>:   prend le chemin et passe d'un repertoire a un autre.                                          |\n");
printf("\t=============================================================================================================\n");
printf("\t| copier <nom> <path>:    copie un fichier dans le path indique                                             |\n");
printf("\t=============================================================================================================\n");
printf("\t| supprimer <nome>:supprime un fichier ou bien repertoire vide !!                                           |\n");
printf("\t=============================================================================================================\n");
printf("\t| supprimer -f <nom>:    supprime recursivement un repertoire et tous ses descendants.                      |\n");
printf("\t=============================================================================================================\n");
printf("\t|NOTE:cette commande  affichera une erreur si la ressource specifiee n'existe pas.                          |\n");
printf("\t=============================================================================================================\n");
printf("\t| rechercher <nom>:    recherche toutes les element portant le nom specifie dans le systeme de fichiers.    |\n");
printf("\t=============================================================================================================\n");
printf("\t|renommer <oldname><newname>:     pour renommer un fichier et le donner le newname                          |\n");
printf("\t=============================================================================================================\n");
printf("\t| lister:    liste tous les elements du repertoire courant.Avec d'autre parametre -D/-F/..                 |\n");
printf("\t=============================================================================================================\n");
printf("\t| chemin:    affiche le chemin absolu vers le repertoire courant.                                           |\n");
printf("\t=============================================================================================================\n");
printf("\t| help --: si vous avez un probleme tapez help -- !!                                                        |\n");
printf("\t=============================================================================================================\n");
printf("\t|NOTE: Si vous tapez help <nome du commande> vous aurez les information necessaire !!                       |\n");
printf("\t=============================================================================================================\n");
printf("\t| quitter:    termine l'execution du programme de gestion du systeme de fichiers.                           |\n");
printf("\t=============================================================================================================\n");


                    printf("\nAppuyer sur une touche pour revenir au menu\n");
                    getch();
                    break;
            case '3' : system ("cls");
```

```
                    printf("\nAppuyer sur une touche pour revenir au menu\n");
                    getch();
                    break;
            case '3' : system ("cls");
                     system ("Color 79");
                    printf("\t\t\t\t\t_____________________________________________\n");
                    printf("\t\t\t\t\t||                                           ||\n");
                    printf("\t\t\t\t\t||       Quitter l'application               ||\n");
                    printf("\t\t\t\t\t||                                           ||\n");
                    printf("\t\t\t\t\t===========================================\n");
                exit(0);
                 break;

            default:
                    {
                     printf("numero de menu invalide\n");
                     Sleep(500);
                    }
        }
}while(1==1);

}
```

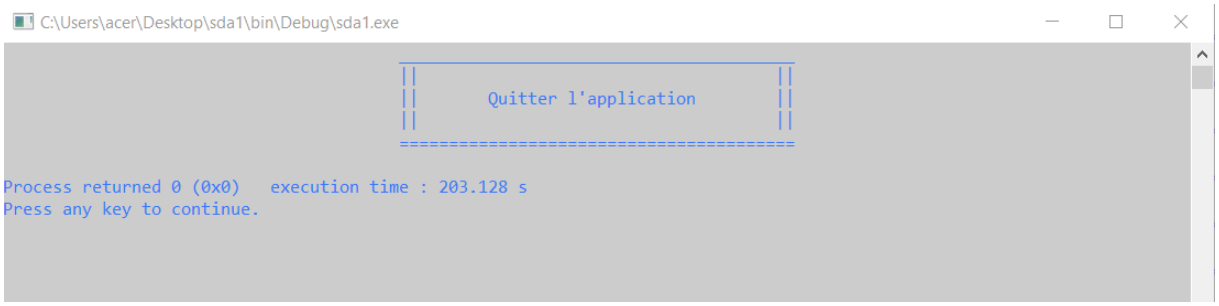
Exécution du programme :

>>L'interface

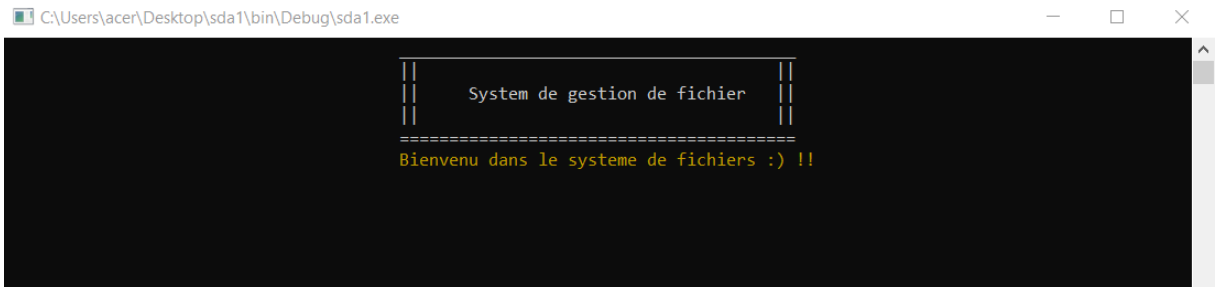
Selon le choix : Si on tape 2 :



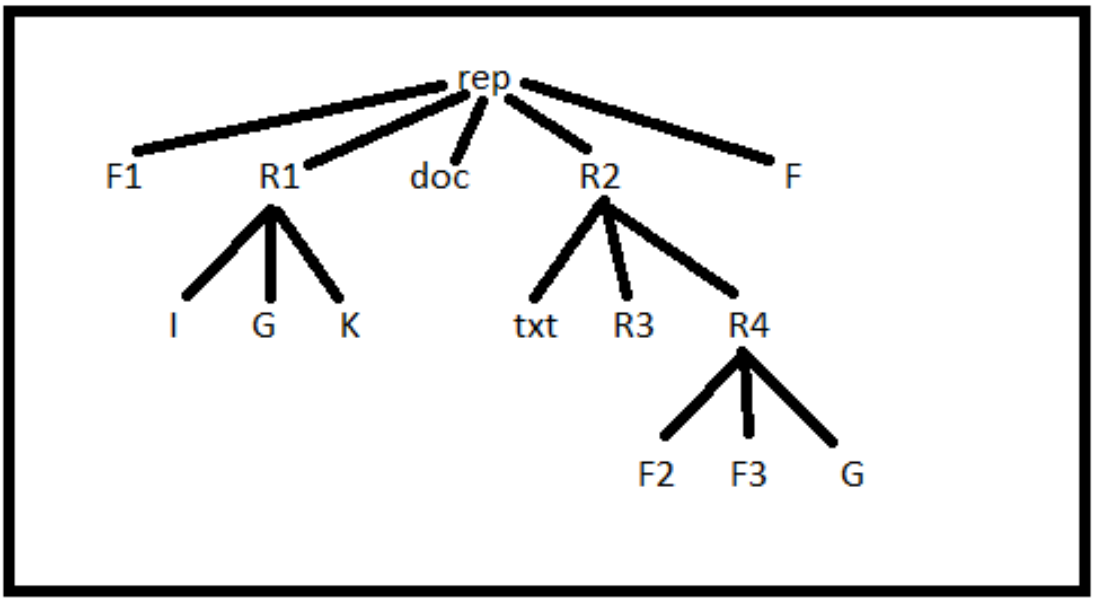
Si on tape 3 :



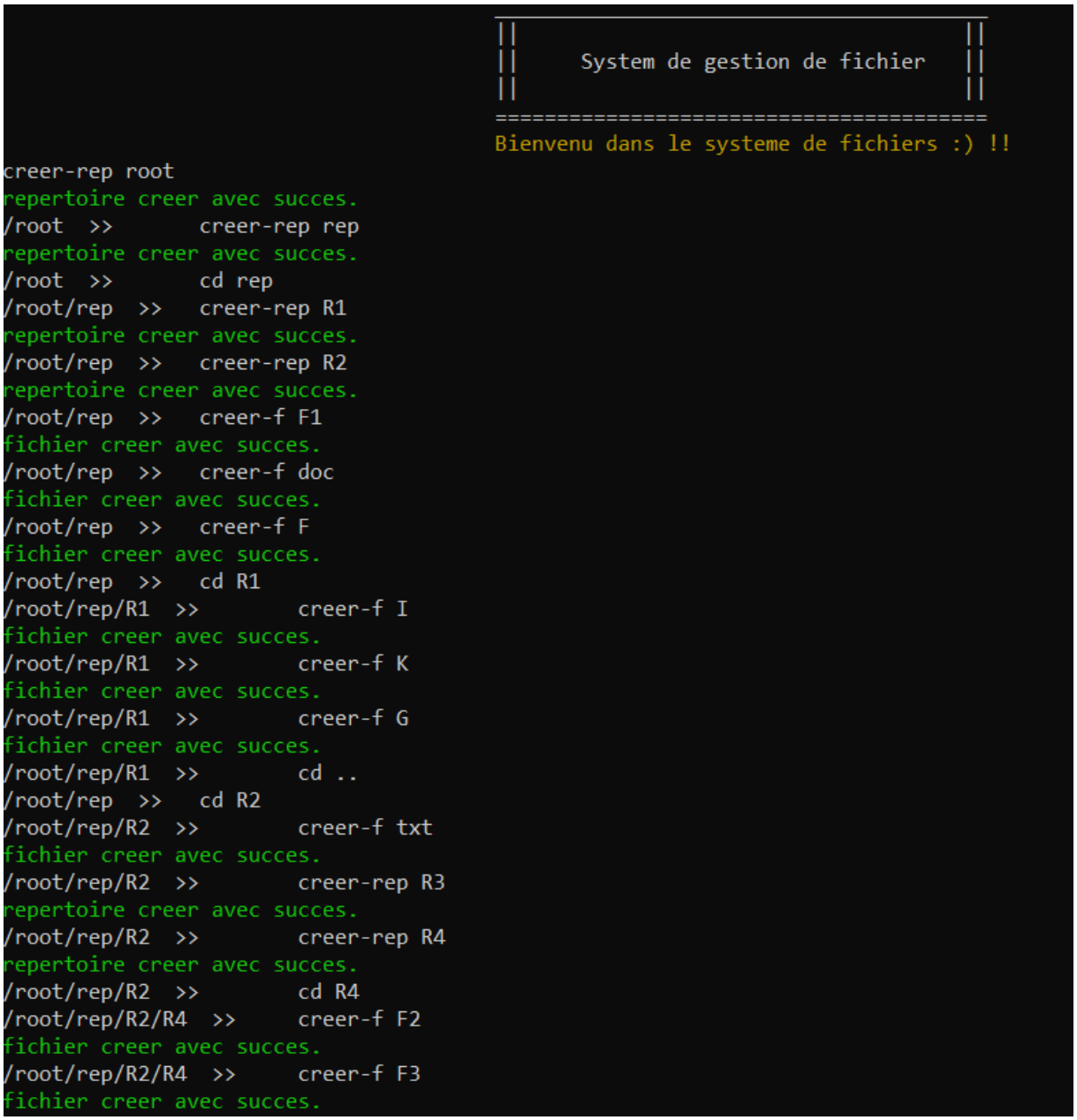
Si on tape 1 :



Pour l'exécution on va travailler sur l'arbre suivante :



>>Création :


System de gestion de fichier
Bienvenu dans le systeme de fichiers :) !!
creer-rep root
repertoire creer avec succes.
/root >> creer-rep rep
repertoire creer avec succes.
/root >> cd rep
/root/rep >> creer-rep R1
repertoire creer avec succes.
/root/rep >> creer-rep R2
repertoire creer avec succes.
/root/rep >> creer-f F1
fichier creer avec succes.
/root/rep >> creer-f doc
fichier creer avec succes.
/root/rep >> creer-f F
fichier creer avec succes.
/root/rep >> cd R1
/root/rep/R1 >> creer-f I
fichier creer avec succes.
/root/rep/R1 >> creer-f K
fichier creer avec succes.
/root/rep/R1 >> creer-f G
fichier creer avec succes.
/root/rep/R1 >> cd ..
/root/rep >> cd R2
/root/rep/R2 >> creer-f txt
fichier creer avec succes.
/root/rep/R2 >> creer-rep R3
repertoire creer avec succes.
/root/rep/R2 >> creer-rep R4
repertoire creer avec succes.
/root/rep/R2 >> cd R4
/root/rep/R2/R4 >> creer-f F2
fichier creer avec succes.
/root/rep/R2/R4 >> creer-f F3
fichier creer avec succes.

>>Lister les répertoires et les fichiers :

```
/root/rep/R2/R4  >>     lister
Listing For '/root/rep/R2/R4':
F F3
F F2
/root/rep/R2/R4  >>     cd ..
/root/rep/R2  >>        lister
Listing For '/root/rep/R2':
D R4
D R3
F txt
/root/rep/R2  >>        cd ..
/root/rep  >>   lister
Listing For '/root/rep':
D R2
D R1
F F
F F1
F doc
/root/rep  >>   cd R1
/root/rep/R1  >>        lister
Listing For '/root/rep/R1':
F G
F K
F I
/root/rep/R1  >>
```

>>Copier le fichier G dans le répertoire R4 :

```
/root/rep/R1  >>        lister
Listing For '/root/rep/R1':
F G
F K
F I
/root/rep/R1  >>        copier G /root/rep/R2/R4
fichier a ete copie avec succes.
/root/rep/R1  >>        lister root/rep/R2/R4
lister les elements pour : '/root':
F G
F F3
F F2
/root/rep/R1  >>
```

>>Lister répertoire/Lister fichier :

```
/root/rep/R1  >>        cd ..
/root/rep  >>   lister -D
impossible de lister les elements du repertoire.
/root/rep  >>   lister-D
Listing For '/root/rep':
D R2
D R1
/root/rep  >>   lister-F
Listing For '/root/rep':
F F
F F1
F doc
/root/rep  >>
```

>>Rechercher :

Le cas où le répertoire ou le fichier existe il affiche le chemin absolue ,sinon un message d'erreur s'affiche.

```
/root/rep  >>   rechercher G
A la recherche de 'G':
 [redacted]      (s'il exist vous aurez tout les chemins absolus de cet element :>; sinon vous aurez rien :<)
F /root/rep/R2/R4/G
F /root/rep/R1/G
/root/rep  >>   rechercher R1
A la recherche de 'R1':
 [redacted]      (s'il exist vous aurez tout les chemins absolus de cet element :>; sinon vous aurez rien :<)
D /root/rep/R1
/root/rep  >>   rechercher R
A la recherche de 'R':
 [redacted]      (s'il exist vous aurez tout les chemins absolus de cet element :>; sinon vous aurez rien :<)
D /root/rep/R2
D /root/rep/R2/R4
D /root/rep/R2/R3
D /root/rep/R1
/root/rep  >>   rechercher X
A la recherche de 'X':
 [redacted]      (s'il exist vous aurez tout les chemins absolus de cet element :>; sinon vous aurez rien :<)
/root/rep  >>
```

>>Renommer :

Renommer le fichier G du répertoire R4 en J

```
/root/rep  >>   cd R2
/root/rep/R2  >>        cd R4
/root/rep/R2/R4  >>     lister
Listing For '/root/rep/R2/R4':
F G
F F3
F F2
/root/rep/R2/R4  >>     renommer G J
fichier renommer avec succes.
/root/rep/R2/R4  >>     lister
Listing For '/root/rep/R2/R4':
F J
F F3
F F2
/root/rep/R2/R4  >>
```

>>Supprimer fichier :

```
/root/rep/R2/R4  >>     lister
Listing For '/root/rep/R2/R4':
F J
F F3
F F2
/root/rep/R2/R4  >>     supprimer J
/root/rep/R2/R4  >>     lister
Listing For '/root/rep/R2/R4':
F F3
F F2
/root/rep/R2/R4  >>
```

>>Supprimer répertoire vide :

```
/root/rep/R2/R4  >>     cd ..
/root/rep/R2  >>        lister
Listing For '/root/rep/R2':
D R4
D R3
F txt
/root/rep/R2  >>        creer-rep R5
repertoire creer avec succes.
/root/rep/R2  >>        lister]
La Command lister] est introuvable :( pour plus d'information tapez help -- !!
/root/rep/R2  >>        lister
Listing For '/root/rep/R2':
D R5
D R4
D R3
F txt
/root/rep/R2  >>        supprimer R5
/root/rep/R2  >>        lister
Listing For '/root/rep/R2':
D R4
D R3
F txt
/root/rep/R2  >>
```

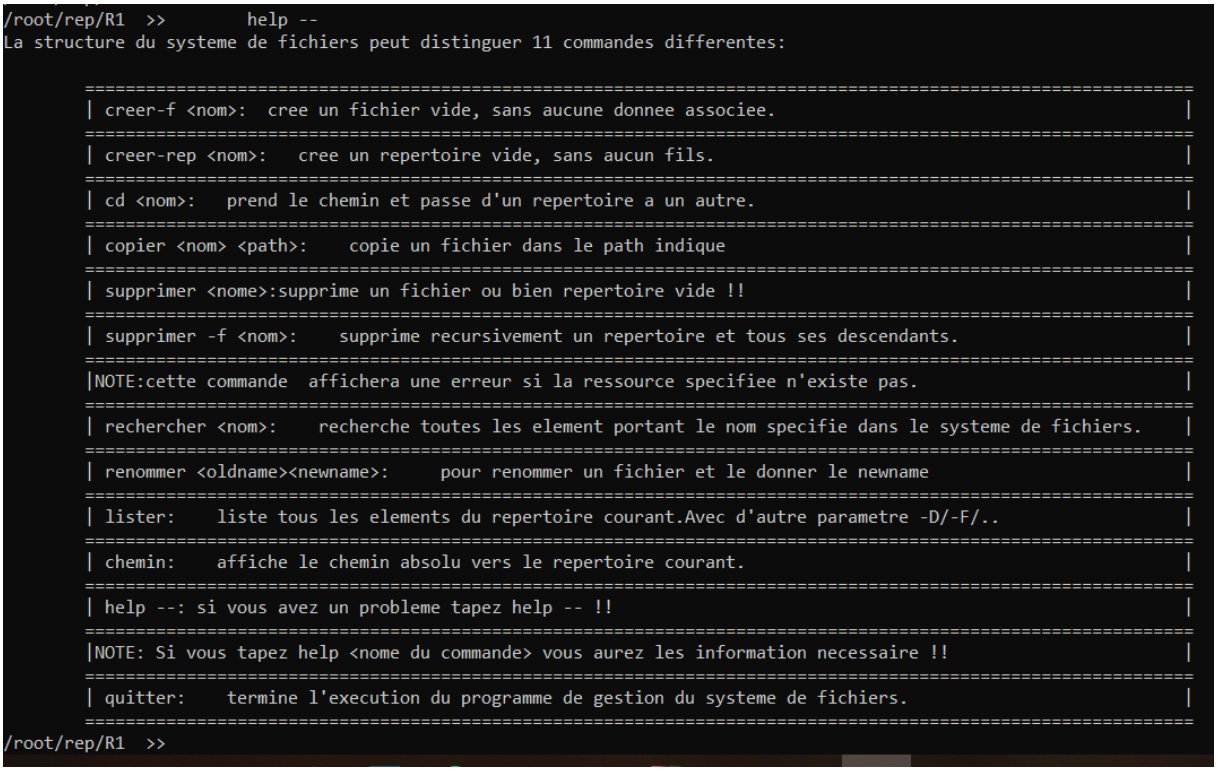
>>Supprimer répertoire non vide :

```
/root/rep/R2/R4  >>     lister ..
lister les elements pour : '/..':
D R4
D R3
F txt
/root/rep/R2  >>        supprimer -f R4
/root/rep/R2  >>
```

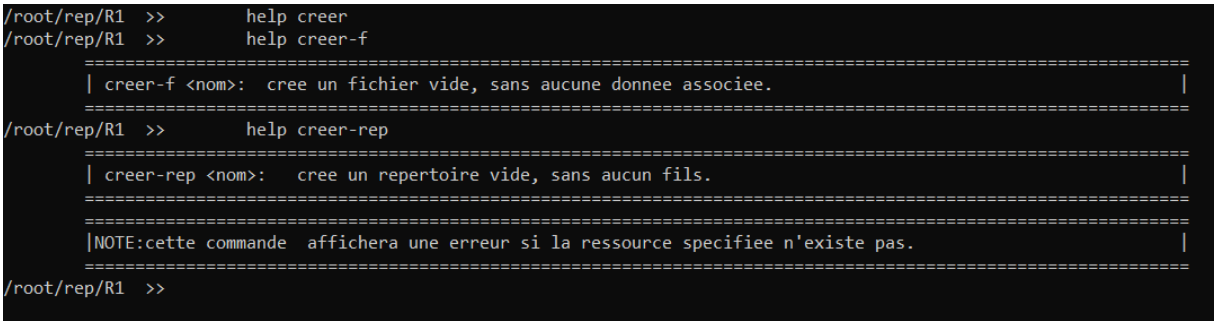
>>Afficher le chemin absolu :

```
/root/rep/R1  >>        cd ..
/root/rep  >>   cd R1
/root/rep/R1  >>        chemin
/root/rep/R1
/root/rep/R1  >>
```

>>Afficher les fonctions du système :



>>Afficher l'information sur la commande :



>>Créer fichier/répertoire avec un nom déjà existant :

```
/root/rep/R1  >>        lister
Listing For '/root/rep/R1':
F G
F K
F I
/root/rep/R1  >>        creer-f K
impossible de creer le fichier.
/root/rep/R1  >>        cd ..
/root/rep  >>   creer-rep R1
```

```
/root/rep  >>   lister
Listing For '/root/rep':
D R2
D R1
/root/rep  >>   creer-rep R2
impossible de creer le repertoire.
/root/rep  >>
```

La commamde ‘quitter ‘ pemet de quitter automatiquemet le gestionnaire de fichiers.

Conclusion

La réalisation de ce projet nous a permis de mettre en pratique nos connaissances acquises au cours du structures de données avancées en général et les arbres en particulier .