Rapport Complet sur system de Gestion de Fichiers

.Table des Matières :

.1 Introduction

Ce projet implémente un système de gestion de fichiers en langage C, permettant de gérer des fichiers avec des enregistrements et d'effectuer diverses opérations comme la création, la suppression, la modification, l'affichage, ainsi que la gestion de la mémoire secondaire. Ce système inclut des fonctionnalités avancées telles que le compactage de la mémoire, la gestion des métadonnées, et la suppression d'un fichier avec nettoyage des enregistrements.

.2 Objectif du projet

L'objectif est de simuler un gestionnaire de fichiers dans un environnement contrôlé avec un disque virtuel simulant la mémoire secondaire. Ce projet traite les concepts de gestion de la mémoire, de manipulation des fichiers et des enregistrements, et d'optimisation de la mémoire via des techniques comme le compactage et la défragmentation.

.3 Structure du projet

.3.1 Structures de données

.1 Enregistrement: L'enregistrement contient un identifiant unique (id) et les données associées à cet enregistrement. Cette structure représente les données qui seront stockées dans les blocs mémoire.

```
10    typedef struct {
11        int id;
12        char data[MAX_DATA_SIZE];
13    } Enregistrement;
```

.2 Bloc: Chaque bloc représente une unité de mémoire dans laquelle un ou plusieurs enregistrements peuvent être stockés. Le bloc peut être libre ou occupé, et contient des informations telles que le nom du fichier, le nombre d'enregistrements et un pointeur vers le bloc suivant.

```
15    typedef struct {
16        int is_free;
17        char fichier_nom[50];
18        int next_bloc;
19        int nb_enregistrements;
20        Enregistrement enreg;
21    } Bloc;
```

3. Fichier: Le fichier est une collection de blocs. Il contient des informations sur le nom du fichier, le nombre d'enregistrements qu'il contient, ainsi que des détails sur son organisation globale et interne.

```
23    typedef struct {
24        char nom[50];
25        int nb_enregistrements;
26        int organisation_globale;
27        int organisation_interne;
28        Bloc *blocs[MAX_DATA_SIZE];
29    } Fichier;
```

4. Metadata: Les métadonnées décrivent la structure globale et interne du fichier ainsi que l'adresse du premier bloc. Elles sont utilisées pour l'accès rapide aux informations sur un fichier spécifique.

```
31    typedef struct {
32        char nom_fichier[50];
33        int taille_fichier_blocs;
34        int taille_fichier_enregs
35        int adresse_premier_bloc;
36        int mode_org_globale;
37        int mode_org_interne;
38    } Metadata;
```

3.2. Fonctions

Le programme contient un ensemble de fonctions permettant de gérer les fichiers, les enregistrements, et la mémoire

1.Gestion de la Mémoire

- initialiser_disque
- mettre_a_jour_etat_blocs

2. Opérations de Fichier

- creer_fichier
- afficher_fichier
- inserer_enregistrement
- rechercher_enregistrement
- modifier_enregistrement
- supprimer_enregistrement
- trier_enregistrements
- defragmenter_fichier
- creer_metadata

3. Gestion de la Mémoire et des Blocs

- compactage
- vider_MS
- verifier_espace_disponible
- recherche_binaire

4. Opérations sur le Fichier

- renommer_fichier
- afficher_metadonnees
- supprimer_fichier
- choisir_fichier
- AfficherEtatMemoire

.4 Analyse détaillée du code

-Fonction : initialiser_disque

```
43  void initialiser_disque(int nb_blocs, int taille_bloc) {
44      memoire = (Bloc *)malloc(nb_blocs * sizeof(Bloc));
45      if (memoire == NULL) {
46          printf("Erreur de memoire !\n");
47          exit(1);
48      }
49      for (int i = 0; i < nb_blocs; i++) {
50          memoire[i].is_free = 1;
51          memoire[i].next_bloc = -1;
52      }
53  }
```

Description :

Cette fonction alloue dynamiquement un espace mémoire structuré en blocs pour simuler un disque virtuel.

Fonctionnement :

.1 Allocation mémoire : Réserve un espace pour le nombre de blocs spécifié. En cas d'échec, le programme affiche une erreur et se termine.

.2 Initialisation des blocs : Chaque bloc est marqué comme libre (is_free = 1) et n'a pas de lien vers un autre bloc (next_bloc = -1).

Utilité :

Préparer un disque virtuel pour la gestion de mémoire ou la simulation de systèmes de fichiers.

Fonction : creer_fichier

```
55  void creer_fichier(Fichier *fichier) {
56      printf("Entrez le nom du fichier: ");
57      scanf("%s", fichier->nom);
58
59      printf("Entrez le nombre d'enregistrements: ");
60      scanf("%d", &fichier->nb_enregistrements);
61
62      printf("Choisissez l'organisation globale (0 pour Contigue, 1 pour Chainee): ");
63      scanf("%d", &fichier->organisation_globale);
64
65      printf("Choisissez l'organisation interne (0 pour Non Triee, 1 pour Triee): ");
66      scanf("%d", &fichier->organisation_interne);
67
68      for (int i = 0; i < fichier->nb_enregistrements; i++) {
69          fichier->blocs[i] = NULL;
70      }
71  }
```

Description :

Cette fonction permet de créer un fichier virtuel en configurant ses propriétés principales, notamment son nom, son nombre d'enregistrements, et ses organisations globale et interne.

Fonctionnement : .1 Saisie des informations : • Nom du fichier (fichier->nom). • Nombre d'enregistrements (fichier->nb_enregistrements). • Organisation globale (contiguë ou chaînée). • Organisation interne (triée ou non triée).

.2 Initialisation des blocs : • Les blocs du fichier sont initialisés à NULL pour indiquer qu'ils ne contiennent aucune donnée au départ.

Utilité :

Préparer la structure d'un fichier virtuel avant son utilisation dans un système de gestion de fichiers.

Fonction : afficher_fichier

```
73  void afficher_fichier(Fichier *fichier) {
74      printf("Nom du fichier: %s\n", fichier->nom);
75      printf("Nombre d'enregistrements: %d\n", fichier->nb_enregistrements);
76      for (int i = 0; i < fichier->nb_enregistrements; i++) {
77          if (fichier->blocs[i] != NULL) {
78              printf("ID: %d, Donnees: %s\n", fichier->blocs[i]->enreg.id, fichier->bloc
79          }
80      }
81  }
```

Description :

Cette fonction affiche les détails d'un fichier virtuel, y compris son nom, le nombre d'enregistrements, et les données contenues dans ses blocs.

Fonctionnement : .1 Affichage des propriétés générales : • Le nom du fichier (fichier->nom). • Le nombre total d'enregistrements (fichier->nb_enregistrements).

.2 Parcours et affichage des blocs : • Pour chaque bloc, si le bloc contient des données (non NULL), affiche l'identifiant de l'enregistrement (ID) et ses données (data).

Utilité :

Permet de visualiser rapidement le contenu et les métadonnées d'un fichier virtuel.

Fonction : inserer_enregistrement

```c
void inserer_enregistrement(Fichier *fichier, Enregistrement enreg) {
    if (fichier->organisation_globale == 0) {
        for (int i = 0; i < fichier->nb_enregistrements; i++) {
            if (fichier->blocs[i] == NULL) {
                fichier->blocs[i] = &memoire[i];
                fichier->blocs[i]->enreg = enreg;
                fichier->blocs[i]->is_free = 0;
                break;
            }
        }
    } else {
        for (int i = 0; i < fichier->nb_enregistrements; i++) {
            if (fichier->blocs[i] == NULL) {
                fichier->blocs[i] = &memoire[i];
                fichier->blocs[i]->enreg = enreg;
                fichier->blocs[i]->is_free = 0;
                if (i < fichier->nb_enregistrements - 1) {
                    fichier->blocs[i]->next_bloc = i + 1;
                } else {
                    fichier->blocs[i]->next_bloc = -1;
                }
                break;
            }
        }
    }
}
```

Description :

Cette fonction insère un enregistrement dans un fichier virtuel selon son organisation globale (contiguë ou chaînée).

Fonctionnement : .1 Vérification de l'organisation globale : • Organisation contiguë (0) : • Parcourt les blocs pour trouver un emplacement libre (NULL). • Insère l'enregistrement dans le premier bloc libre. • Organisation chaînée (1) : • Parcourt les blocs pour trouver un emplacement libre. • Insère l'enregistrement dans le bloc libre et met à jour l'attribut next_bloc pour maintenir la chaîne..2 Mise à jour des propriétés du bloc : • Le bloc est associé à la mémoire (memoire[i]). • Les données de l'enregistrement sont copiées dans le bloc. • L'attribut is_free est défini à 0 pour indiquer que le bloc est occupé.

Utilité :

Gérer l'insertion d'enregistrements dans un fichier virtuel selon des stratégies d'organisation spécifiques.

Fonction : rechercher_enregistrement

```c
void rechercher_enregistrement(Fichier *fichier, int id) {
    for (int i = 0; i < fichier->nb_enregistrements; i++) {
        if (fichier->blocs[i] != NULL && fichier->blocs[i]->enreg.id == id) {
            printf("Enregistrement trouve: ID = %d, Donnees = %s\n", fichier->blocs[
            return;
        }
    }
    printf("Enregistrement non trouve.\n");
}
```

Description :

Cette fonction permet de rechercher un enregistrement spécifique dans un fichier virtuel à l'aide de son identifiant (id).

Fonctionnement : .1 Parcours des blocs : • Parcourt tous les blocs du fichier virtuel. • Vérifie si le bloc n'est pas vide (NULL) et si l'identifiant de l'enregistrement correspond à l'identifiant recherché.

.2 Affichage du résultat : • Si l'enregistrement est trouvé, affiche ses détails (ID et données). • Si aucun enregistrement ne correspond, affiche un message indiquant qu'il n'a pas été trouvé.

Utilité :

Facilite l'accès direct à un enregistrement dans un fichier virtuel en fonction de son identifiant unique.

Fonction : modifier_enregistrement

```c
void modifier_enregistrement(Fichier *fichier, int id, char *new_data) {
    for (int i = 0; i < fichier->nb_enregistrements; i++) {
        if (fichier->blocs[i] != NULL && fichier->blocs[i]->enreg.id == id) {
            strcpy(fichier->blocs[i]->enreg.data, new_data);
            printf("Enregistrement modifie: ID = %d, Donnees = %s\n", fichier->blocs
            return;
        }
    }
    printf("Enregistrement non trouve.\n");
}
```

Description :

Cette fonction modifie les données d'un enregistrement existant dans un fichier virtuel en utilisant son identifiant (id).

Fonctionnement : .1 Parcours des blocs : • Parcourt tous les blocs du fichier virtuel. • Vérifie si le bloc n'est pas vide (NULL) et si l'identifiant de l'enregistrement correspond à celui fourni.

.2 Modification des données : • Remplace les données actuelles de l'enregistrement par les nouvelles données (new_data). .3 Affichage du résultat : • Si l'enregistrement est trouvé, affiche un message confirmant la modification avec les nouvelles données. • Si aucun enregistrement ne correspond, affiche un message d'erreur.

Utilité :

Permet de mettre à jour les informations stockées dans un enregistrement spécifique d'un fichier virtuel.

Fonction : supprimer_enregistrement

```c
void supprimer_enregistrement(Fichier *fichier, int id) {
    for (int i = 0; i < fichier->nb_enregistrements; i++) {
        if (fichier->blocs[i] != NULL && fichier->blocs[i]->enreg.id == id) {
            fichier->blocs[i]->is_free = 1;
            fichier->blocs[i] = NULL;
            printf("Enregistrement supprime: ID = %d\n", id);
            return;
        }
    }
    printf("Enregistrement non trouve.\n");
}
```

Description :

Cette fonction supprime un enregistrement dans un fichier virtuel en le recherchant par son identifiant (id) et en libérant son bloc associé.

Fonctionnement : .1 Parcours des blocs : • Parcourt tous les blocs du fichier virtuel. • Vérifie si le bloc contient un enregistrement avec l'identifiant spécifié.

.2 Suppression de l'enregistrement : • Marque le bloc comme libre (is_free = 1). • Réinitialise le pointeur du bloc à NULL pour indiquer qu'il ne contient plus de données.

.3 Affichage du résultat : • Si l'enregistrement est trouvé, affiche un message confirmant la suppression. • Sinon, affiche un message indiquant que l'enregistrement n'a pas été trouvé.

Utilité :

Libérer l'espace mémoire occupé par un enregistrement spécifique dans un fichier virtuel.

Fonction : trier_enregistrements

```
143  void trier_enregistrements(Fichier *fichier) {
144      for (int i = 0; i < fichier->nb_enregistrements - 1; i++) {
145          for (int j = i + 1; j < fichier->nb_enregistrements; j++) {
146              if (fichier->blocs[i] != NULL && fichier->blocs[j] != NULL &&
147                  fichier->blocs[i]->enreg.id > fichier->blocs[j]->enreg.id) {
148                  Bloc *temp = fichier->blocs[i];
149                  fichier->blocs[i] = fichier->blocs[j];
150                  fichier->blocs[j] = temp;
151              }
152          }
153      }
154      printf("Les enregistrements ont ete tries.\n");
155  }
```

Description :

Cette fonction trie les enregistrements d'un fichier virtuel en ordre croissant selon leurs identifiants (id).

Fonctionnement : .1 Tri par comparaison : • Utilise un tri à bulles modifié pour comparer les identifiants des enregistrements de chaque bloc. • Échange les blocs si un enregistrement avec un identifiant plus grand précède un autre avec un identifiant plus petit.

.2 Gestion des blocs vides : • Ignore les blocs vides (NULL) lors des comparaisons et des échanges.

.3 Affichage du résultat : • Affiche un message confirmant que les enregistrements ont été triés.

Utilité :

Réorganiser les enregistrements d'un fichier virtuel pour faciliter les recherches ou améliorer l'efficacité des opérations.

Fonction : defragmenter_fichier

```
157  void defragmenter_fichier(Fichier *fichier) {
158      int index = 0;
159      for (int i = 0; i < fichier->nb_enregistrements; i++) {
160          if (fichier->blocs[i] != NULL) {
161              if (i != index) {
162                  fichier->blocs[index] = fichier->blocs[i];
163                  fichier->blocs[i] = NULL;
164              }
165              index++;
166          }
167      }
168      printf("Le fichier a ete defragmente.\n");
169  }
```

Description :

Cette fonction réorganise les blocs d'un fichier virtuel en déplaçant les blocs occupés vers le début du tableau et en libérant les espaces vides.

Fonctionnement : .1 Réorganisation des blocs : • Parcourt tous les blocs du fichier. • Déplace les blocs non vides vers le début du tableau pour éliminer les espaces vides entre les blocs. • Réinitialise les positions des blocs vides (NULL) après chaque déplacement.

2. Mise à jour de l'index : • L'index est incrémenté chaque fois qu'un bloc non vide est déplacé, garantissant ainsi que les blocs occupés restent dans l'ordre.

.3 Affichage du résultat : • Affiche un message confirmant que le fichier a été défragmenté.

Utilité :

Optimiser l'organisation des blocs d'un fichier virtuel en regroupant les blocs occupés et en réduisant l'espace mémoire fragmenté.

Fonction : creer_metadata

```
171  void creer_metadata(Fichier *fichier, Metadata *metadata) {
172      strcpy(metadata->nom_fichier, fichier->nom);
173      metadata->taille_fichier_blocs = fichier->nb_enregistrements;
174      metadata->taille_fichier_enregs = fichier->nb_enregistrements;
175      metadata->adresse_premier_bloc = (fichier->blocs[0] != NULL) ? fichier->blocs[0]
176      metadata->mode_org_globale = fichier->organisation_globale;
177      metadata->mode_org_interne = fichier->organisation_interne;
178  }
```

Description :

Cette fonction génère des métadonnées pour un fichier virtuel, contenant des informations essentielles sur sa structure et son organisation.

Fonctionnement : .1 Initialisation des métadonnées : • Le nom du fichier est copié dans les métadonnées (metadata->nom_fichier). • La taille du fichier en termes de nombre de blocs et d'enregistrements est définie. • L'adresse du premier bloc est calculée en utilisant un pointeur relatif par rapport à la mémoire, ou définie à -1 si le premier bloc est NULL. • Les modes d'organisation globale et interne sont copiés depuis le fichier.

.2 Mise à jour des champs des métadonnées : • taille_fichier_blocs et taille_fichier_enregs sont tous deux définis en fonction du nombre d'enregistrements du fichier. • mode_org_globale et mode_org_interne sont définis selon les paramètres du fichier.

Utilité :

Permet de créer un ensemble de métadonnées pour un fichier virtuel, fournissant des informations essentielles pour son gestionnaire de système ou pour des opérations ultérieures de manipulation ou de gestion.

Fonction : recherche_binaire

```c
int recherche_binaire(Fichier *fichier, int id) {
    int gauche = 0;
    int droite = fichier->nb_enregistrements - 1;

    while (gauche <= droite) {
        int milieu = (gauche + droite) / 2;
        if (fichier->blocs[milieu] != NULL && fichier->blocs[milieu]->enreg.id == id) {
            printf("Enregistrement trouve au bloc %d\n", milieu);
            return milieu;
        } else if (fichier->blocs[milieu] != NULL && fichier->blocs[milieu]->enreg.id <
            gauche = milieu + 1;
        } else {
            droite = milieu - 1;
        }
    }

    printf("Enregistrement non trouve.\n");
    return -1;
}
```

Description :

Cette fonction implémente l'algorithme de recherche binaire pour trouver un enregistrement dans un fichier virtuel, en utilisant l'identifiant (id) de l'enregistrement. Elle nécessite que les enregistrements soient triés.

Fonctionnement : .1 Initialisation des indices : • Déclare deux indices : gauche (début du tableau) et droite (fin du tableau

..2 Algorithme de recherche binaire : • Tant que l'intervalle de recherche est valide (gauche <= droite), calcule l'indice du milieu. • Si l'enregistrement au milieu correspond à l'identifiant recherché, retourne l'indice du bloc. • Si l'enregistrement au milieu a un identifiant inférieur, cherche dans la moitié droite en déplaçant l'indice gauche. • Si l'enregistrement au milieu a un identifiant supérieur, cherche dans la moitié gauche en déplaçant l'indice droite..3 Affichage du résultat : • Si l'enregistrement est trouvé, affiche son emplacement dans le fichier. • Si l'enregistrement n'est pas trouvé, affiche un message d'erreur et retourne -1.

Utilité :

Permet de rechercher rapidement un enregistrement dans un fichier virtuel trié, avec une complexité logarithmique (O(log n)), idéale pour les fichiers volumineux.

Fonction : verifier_espace_disponible

Description :

```
200  int verifier_espace_disponible(Fichier *fichier) {
201      int espaces_libres = 0;
202      for (int i = 0; i < MAX_DATA_SIZE; i++) {
203          if (memoire[i].is_free == 1) {
204              espaces_libres++;
205          }
206      }
207
208      if (espaces_libres >= fichier->nb_enregistrements) {
209          return 1;
210      } else {
211          printf("Espace insuffisant, veuillez effectuer un compactage.\n");
212          return 0;
```

Cette fonction vérifie si suffisamment d'espace mémoire est disponible pour un fichier virtuel, en fonction du nombre d'enregistrements qu'il contient.

Fonctionnement : .1 Calcul du nombre d'espaces libres : • Parcourt la mémoire pour compter le nombre de blocs libres (celles dont is_free == 1)

. .2 Vérification de l'espace disponible : • Si le nombre d'espaces libres est supérieur ou égal au nombre d'enregistrements requis pour le fichier (fichier->nb_enregistrements), retourne 1 pour indiquer qu'il y a suffisamment d'espace. • Si l'espace est insuffisant, affiche un message d'avertissement et retourne 0.

Utilité :

Permet de vérifier si la mémoire disponible est suffisante pour stocker de nouveaux enregistrements dans un fichier virtuel, et de signaler si un compactage est nécessaire.

Fonction : mettre_a_jour_etat_blocs

```
216  void mettre_a_jour_etat_blocs() {
217      for (int i = 0; i < MAX_DATA_SIZE; i++) {
218          if (memoire[i].is_free == 0 && memoire[i].next_bloc == -1) {
219              memoire[i].is_free = 1;
220          }
221      }
222  }
```

Description :

Cette fonction met à jour l'état des blocs dans la mémoire en libérant ceux qui sont occupés mais n'ont pas de bloc suivant (next_bloc == -1).

Fonctionnement : .1 Parcours des blocs de mémoire : • La fonction parcourt tous les blocs de mémoire (jusqu'à MAX_DATA_SIZE).

.2 Vérification et mise à jour de l'état : • Pour chaque bloc occupé (is_free == 0) et dont l'attribut next_bloc est égal à -1 (indiquant qu'il n'y a pas de bloc suivant), l'état de ce bloc est mis à jour pour le marquer comme libre (is_free = 1).

Utilité :

Permet de libérer de l'espace dans la mémoire en réinitialisant les blocs qui ne sont plus utilisés, afin de les rendre disponibles pour de futurs enregistrements.

Fonction : vider_MS

```
224  void vider_MS() {
225      for (int i = 0; i < MAX_DATA_SIZE; i++) {
226          memoire[i].is_free = 1;
227          memoire[i].next_bloc = -1;
228      }
229  }
```

Description :

Cette fonction vide toute la mémoire simulée en réinitialisant l'état de chaque bloc à "libre" et en réinitialisant les pointeurs associés.

Fonctionnement : .1 Parcours des blocs de mémoire : • La fonction parcourt tous les blocs de mémoire (jusqu'à MAX_DATA_SIZE).

.2 Réinitialisation des blocs : • Chaque bloc est marqué comme libre (is_free = 1). • Le pointeur vers le bloc suivant est réinitialisé à -1 (next_bloc = -1), indiquant qu'il n'y a plus de lien vers un autre bloc.

Utilité :

Permet de vider l'ensemble de la mémoire simulée, libérant ainsi tout l'espace de stockage et réinitialisant l'organisation des blocs pour de nouvelles opérations.

Fonction : compactage

```
230  void compactage() {
231      int i = 0;
232      for (int j = 0; j < MAX_DATA_SIZE; j++) {
233          if (memoire[j].is_free == 0) {
234              if (i != j) {
235                  memoire[i] = memoire[j];
236                  memoire[j].is_free = 1;
237              }
238              i++;
239          }
240      }
241      printf("Compactage effectue.\n");
242  }
```

Description :

Cette fonction effectue un compactage de la mémoire simulée en réorganisant les blocs occupés pour réduire les espaces vides, ce qui améliore l'utilisation de la mémoire.

Fonctionnement : .1 Parcours des blocs de mémoire : • La fonction parcourt tous les blocs de mémoire (MAX_DATA_SIZE). .2 Déplacement des blocs occupés : • Lorsque la fonction rencontre un bloc occupé (is_free == 0), elle déplace ce bloc vers l'avant (vers l'indice i) pour combler les espaces vides. • Après le déplacement, elle marque le bloc d'origine comme libre (is_free = 1). .3 Affichage du résultat : • Une fois le compactage effectué, la fonction affiche un message indiquant que le compactage a été réalisé.

Utilité :

Optimise l'utilisation de la mémoire en réduisant les fragments et en libérant de l'espace pour de nouveaux enregistrements.

Fonction : renommer_fichier

```
244     void renommer_fichier(Fichier *fichier, char *new_name) {
245         strcpy(fichier->nom, new_name);
246         printf("Le fichier a ete renomme en %s.\n", fichier->nom);
247     }
```

Description :

Cette fonction permet de renommer un fichier virtuel en modifiant son nom.

Fonctionnement

: .1 Mise à jour du nom du fichier : • La fonction copie le nouveau nom (new_name) dans l'attribut nom du fichier virtuel.

.2 Affichage du résultat : • Une fois le fichier renommé, un message est affiché pour indiquer le nouveau nom du fichier.

Utilité :

Permet de modifier facilement le nom d'un fichier virtuel, ce qui peut être utile pour la gestion ou l'organisation des fichiers.

Fonction : afficher_metadonnees

```
249  void afficher_metadonnees(Fichier *fichier) {
250      printf("Metadonnees du fichier %s:\n", fichier->nom);
251      printf("Organisation globale: %d\n", fichier->organisation_globale);
252      printf("Organisation interne: %d\n", fichier->organisation_interne);
253      printf("Nombre d'enregistrements: %d\n", fichier->nb_enregistrements);
254  }
```

Description :

Cette fonction affiche les métadonnées d'un fichier virtuel, permettant de visualiser les informations de base liées à son organisation et à ses enregistrements.

Fonctionnement : .1 Affichage des informations du fichier : • Le nom du fichier est affiché. • L'organisation globale et interne du fichier sont présentées sous forme de valeurs numériques. • Le nombre d'enregistrements du fichier est également affiché.

.2 Affichage des métadonnées : • Ces informations permettent à l'utilisateur de connaître la structure du fichier et ses attributs clés.

Utilité :

Offre une vue d'ensemble rapide des métadonnées d'un fichier virtuel, utile pour vérifier sa configuration et son organisation avant de procéder à des opérations de gestion.

Fonction : supprimer_fichier

```c
void supprimer_fichier(Fichier *fichier) {
    for (int i = 0; i < fichier->nb_enregistrements; i++) {
        if (fichier->blocs[i] != NULL) {
            fichier->blocs[i]->is_free = 1;
            fichier->blocs[i] = NULL;
        }
    }
    printf("Le fichier %s a ete supprime.\n", fichier->nom);
}
```

Description :

Cette fonction permet de supprimer un fichier virtuel en libérant tous les blocs de mémoire qui lui sont associés.

Fonctionnement : .1 Libération des blocs de mémoire : • La fonction parcourt tous les blocs associés au fichier. • Pour chaque bloc qui n'est pas NULL, elle le marque comme libre (is_free = 1) et réinitialise le pointeur vers le bloc à NULL.

.2 Affichage du résultat : • Une fois tous les blocs libérés, un message est affiché indiquant que le fichier a été supprimé.

Utilité :

Permet de supprimer un fichier virtuel et de libérer tous les blocs mémoire associés, en rendant l'espace de stockage disponible pour d'autres fichiers ou enregistrements.

Fonction : choisir_fichier

```c
int choisir_fichier(Fichier *fichiers, int nb_fichiers) {
    int choix;
    printf("\nChoisissez un fichier:\n");
    for (int i = 0; i < nb_fichiers; i++) {
        printf("%d. %s\n", i + 1, fichiers[i].nom);
    }
    printf("Entrez le numero du fichier: ");
    scanf("%d", &choix);
    return choix - 1;
}
```

Description :

Cette fonction permet à l'utilisateur de choisir un fichier parmi une liste de fichiers existants en lui attribuant un numéro.

Fonctionnement : .1 Affichage des fichiers disponibles : • La fonction affiche la liste des fichiers avec un numéro associé à chacun. • La liste est générée à partir d'un tableau de fichiers (fichiers).

.2 Sélection du fichier : • L'utilisateur entre un numéro correspondant à un fichier de la liste. • Le numéro saisi est ajusté pour correspondre à l'indice du fichier dans le tableau (choix - 1), car l'indice dans le tableau commence à partir de 0.

.3 Retour de la sélection : • La fonction retourne l'indice du fichier sélectionné dans le tableau.

Utilité :

Permet à l'utilisateur de sélectionner un fichier à manipuler dans un ensemble de fichiers existants, facilitant ainsi l'interaction avec plusieurs fichiers.

Fonction : afficherEtatMemoire

```c
void afficherEtatMemoire() {
    printf("\nEtat de la memoire secondaire :\n");
    for (int i = 0; i < MAX_DATA_SIZE; i++) {
        printf("----------------------\n");
        printf("| ");
        if (memoire[i].is_free) {
            printf("\033[1;32mLibre\033[0m");
        } else {
            printf("\033[1;31m%-8s %2d enreg\033[0m", memoire[i].fichier_nom, memoire[
        }
        printf(" |\n");
        printf("----------------------\n");
    }
}
```

Description :

Cette fonction affiche l'état actuel de la mémoire secondaire, en montrant si chaque bloc est libre ou occupé, et, dans le cas où il est occupé, elle indique le nom du fichier et le nombre d'enregistrements associés.

Fonctionnement : .1 Parcours des blocs de mémoire : • La fonction parcourt tous les blocs de mémoire (MAX_DATA_SIZE).

.2 Affichage de l'état de chaque bloc : • Si un bloc est libre (is_free == 1), il est affiché en vert et marqué comme "Libre". • Si un bloc est occupé, il est affiché en rouge et montre le nom du fichier (fichier_nom) et le nombre d'enregistrements (nb_enregistrements) associés à ce bloc.

.3 Formatage de l'affichage : • Le bloc est affiché avec des lignes de séparation et des couleurs pour rendre l'information plus lisible.

Utilité :

Permet de visualiser rapidement l'état de la mémoire secondaire, ce qui est utile pour la gestion de l'espace de stockage, afin de savoir quels blocs sont utilisés et quels sont libres.

Résumé final

En résumé, ce projet présente une implémentation simple mais fonctionnelle d'un système de gestion de fichiers en C. Les fonctionnalités implémentées permettent de créer, gérer, et manipuler des fichiers et des enregistrements tout en assurant une gestion efficace de la mémoire.

5. Conclusion

Dans ce projet, nous avons créé un système de gestion de fichiers simulé en C, qui permet de réaliser plusieurs opérations sur des fichiers et leurs enregistrements. L'objectif principal était de concevoir un modèle fonctionnel de gestion de la mémoire, où chaque fichier est stocké dans une "mémoire secondaire" simulée, et où diverses opérations comme la création, la suppression, l'insertion, la recherche et la modification de fichiers et d'enregistrements sont implémentées.

Les fonctionnalités principales du projet incluent : • Création et gestion de fichiers : Le système permet de créer des fichiers, d'y ajouter des enregistrements et d'y effectuer des modifications. • Gestion de la mémoire : Le système gère la mémoire secondaire en utilisant des blocs. Ces blocs peuvent être marqués comme "libres" ou "occupés", et les opérations comme la suppression ou la compactage sont mises en place pour optimiser l'utilisation de la mémoire. • Optimisation de la mémoire (Compactage) : Le système permet de réduire l'espace utilisé en déplaçant les blocs occupés pour éliminer les espaces vides. • Affichage des métadonnées : Le système permet de visualiser les informations concernant les fichiers, comme l'organisation interne et externe des données.

6. Sources

Les sources suivantes ont été utilisées pour la mise en œuvre de ce projet

1. Livre "The C ProgrammingLanguage" de Brian W. Kernighan et DennisM. Ritchie.
2. Chatgpt by OpenAI
3. Cours "Data Structures and Algorithms in C".
4. Article "Memory Management in C" sur GeeksforGeeks.
5. "Operating Systems: Three Easy Pieces" de Remzi H. Arpaci-Dusseau.
6. Articles "File Systems and Storage" sur Techopedia.