Rapport de Projet

Simulation de Systèmes de Gestion de Fichiers (SGF)

Projet réalisé par : **SAHRAOUI Amani**

Étudiante à l'USTHB

2ème Année Ingénieur en Informatique
Groupe 2 - Section C

Encadré par : **Dr. LAHRECHE Abdelmadjid**

January 10, 2025

Ce projet a été réalisé dans le cadre du module **SFSD (Systèmes de Fichiers)**.
Objectifs principaux :

- Simulation de deux SGF en deux couches : système et utilisateur.
- Gestion de la mémoire secondaire en mode contigu et chaîné.
- Création de fichiers, enregistrements et données.
- Menu textuel interactif pour gérer les métadonnées et interfacer avec l'utilisateur.

Université des Sciences et de la Technologie Houari Boumédiène
Département Informatique

Introduction

Les systèmes de fichiers ont joué un rôle central dans l'évolution des systèmes d'exploitation, en permettant l'organisation et la gestion efficace des données stockées sur des supports physiques. Depuis leurs débuts, ces systèmes ont évolué pour répondre aux besoins croissants en termes de performance, de fiabilité et de gestion de la mémoire. Deux méthodes historiques de gestion des fichiers se démarquent : **l'allocation contiguë** et **l'allocation chaînée**.

L'allocation contiguë repose sur l'attribution de blocs consécutifs pour stocker un fichier. Bien que cette méthode soit rapide pour l'accès séquentiel, elle souffre de limitations comme la fragmentation externe et la difficulté de redimensionnement des fichiers. En revanche, l'allocation chaînée utilise des pointeurs pour relier des blocs non contigus. Cela permet une meilleure flexibilité dans l'utilisation de l'espace disque, mais introduit des surcoûts lors de l'accès, en particulier pour les lectures aléatoires. Ces approches, bien qu'anciennes, restent pertinentes dans la compréhension des systèmes modernes.

Dans le cadre de ce projet, un simulateur simplifié de système de gestion de fichiers (SGF) a été développé, appelé **FSMSim**. Le programme implémente plusieurs fonctionnalités permettant de simuler des opérations fondamentales sur des fichiers, telles que l'initialisation de la mémoire secondaire, la création et le chargement de fichiers, la gestion des enregistrements, et des opérations avancées comme la défragmentation et le compactage. Ces fonctionnalités, accessibles via un menu interactif, reflètent les aspects pratiques des systèmes de gestion de fichiers contemporains.

Cependant, la réalisation de ce projet n'a pas été sans défis. Le codage en langage C, notamment sous l'environnement de développement Visual Studio Code, a présenté plusieurs difficultés. Parmi celles-ci figurent des erreurs liées à la gestion de la mémoire dynamique, la complexité de la gestion des structures imbriquées, et des problèmes spécifiques au débogage, particulièrement pour la manipulation des pointeurs et des allocations de mémoire. De plus, l'utilisation des codes ANSI pour styliser l'affichage a parfois nécessité des ajustements pour assurer la compatibilité avec différents terminaux.

Ce rapport détaille les étapes de conception et d'implémentation de ce simulateur, en mettant en lumière les choix techniques réalisés, les défis rencontrés, et les solutions apportées pour créer un outil fonctionnel et éducatif.

1 Avantages et inconvénients des modes contigu et chaîné

La gestion de la mémoire dans un système de fichiers peut être réalisée de différentes manières, dont les deux modes les plus courants sont le mode **contigu** et le mode **chaîné**. Chacun de ces modes présente des avantages et des inconvénients qui influent sur la performance, la gestion des fichiers et la flexibilité du système de fichiers.

1.1 Mode contigu

Dans le mode contigu, les données d'un fichier sont stockées dans un bloc de mémoire contigu, c'est-à-dire que tous les blocs de données du fichier sont adjacents les uns aux autres en mémoire.

Avantages :

- **Accès rapide aux données :** L'un des principaux avantages du mode contigu est que les blocs de données sont situés les uns à côté des autres, ce qui permet un accès

séquentiel rapide aux données. Les têtes de lecture/écriture n'ont pas à se déplacer beaucoup pour accéder aux différentes parties du fichier.

- **Simplicité de gestion :** Le système est simple à mettre en œuvre, car il n'y a pas de liens à gérer entre les blocs. L'adressage est direct, ce qui simplifie l'allocation et la récupération des fichiers.

Inconvénients :

- **Fragmentation externe :** L'un des plus grands inconvénients de ce mode est la fragmentation externe. Avec le temps, les fichiers créés et supprimés peuvent laisser des espaces vides entre les fichiers, rendant difficile l'allocation de blocs contigus pour de nouveaux fichiers ou l'extension de fichiers existants.

- **Peu flexible :** L'allocation de mémoire doit être faite de manière prévisionnelle, et si l'espace contigu nécessaire n'est pas disponible, cela peut entraîner des erreurs ou des rejets d'allocation.

1.2 Mode chaîné

Dans le mode chaîné, les données d'un fichier sont dispersées sur plusieurs blocs, mais chaque bloc contient un pointeur vers le bloc suivant, permettant de retrouver l'ensemble des données du fichier.

Avantages :

- **Absence de fragmentation externe :** Contrairement au mode contigu, le mode chaîné ne souffre pas de fragmentation externe, car chaque bloc peut être alloué indépendamment, même si des espaces vides existent ailleurs en mémoire.

- **Flexibilité :** Ce mode permet une allocation plus flexible de la mémoire, car les blocs peuvent être placés n'importe où dans la mémoire, réduisant ainsi les problèmes d'espace insuffisant pour de nouveaux fichiers.

Inconvénients :

- **Accès plus lent :** L'un des principaux inconvénients est que l'accès aux données peut être plus lent. Pour lire ou écrire un fichier, le système doit suivre les pointeurs de chaque bloc, ce qui peut impliquer plusieurs accès mémoire et ralentir les performances, en particulier pour les fichiers volumineux.

- **Complexité de gestion :** La gestion des pointeurs de chaque bloc ajoute une couche de complexité au système de fichiers. Il faut également gérer les cas où des blocs sont corrompus ou perdus.

Critère	**Mode contigu**	**Mode chaîné**
Gestion de l'espace	Fragmentation externe	Utilisation efficace
Temps d'accès	Accès direct rapide	Accès séquentiel
Complexité	Simple à implémenter	Gestion de pointeurs requise

Table 1: Comparaison des modes contigu et chaîné

2 Objectifs du projet

Ce projet a pour objectif principal de développer un simulateur de systèmes de gestion de fichiers (SGF) permettant d'illustrer les mécanismes d'allocation de mémoire secondaire en mode contigu et chaîné.

2.1 Structure du Projet

Le projet repose sur une organisation modulaire pour garantir lisibilité et maintenabilité :

- `main.c` **:** Ce fichier principal implémente le menu interactif permettant à l'utilisateur de naviguer parmi les fonctionnalités. Il appelle les fonctions nécessaires pour exécuter les choix du menu.
- `functionsContigues.h` **et** `functionsChaine.h` **:** Ces fichiers contiennent respectivement les implémentations des opérations pour les modes contigu et chaîné.
- `structures.h` **:** Définit les structures de données utilisées dans le projet, comme les blocs, les fichiers, et la mémoire.
- `buffer.h` **:** Implémente la gestion du buffer, avec des fonctions pour l'initialisation, la lecture, et l'écriture.

Cette structure modulaire facilite le développement et l'intégration des fonctionnalités tout en séparant clairement les responsabilités des différents composants. Cette structure permet aussi de simplifier la mise en œuvre du système de gestion de fichier dans le système d'exploitation. c'est une simulation qui permet de vérifier des frais fonctions systèmes qui nécessiterons une intégration avec le mode protéger du système en l'occurrence les interruptions ou ce que l'on appel les drivers du disque.

2.2 Fonctions et procédures

Le simulateur offre une interface utilisateur textuelle interactive et comprend les fonctionnalités suivantes :

- **Initialisation de la mémoire secondaire** : La fonction `initialiserMemoireSecondaire` permet de préparer une structure de mémoire secondaire en spécifiant son mode d'allocation.
- **Création de fichiers** : La fonction `creer_FichierContigue` permet d'ajouter un fichier avec un nom et une taille en blocs dans la mémoire secondaire.
- **Affichage des métadonnées** : La fonction `afficher_Metadonnees` permet de visualiser les informations liées aux fichiers, comme leur taille, emplacement et mode d'allocation.
- **Gestion des enregistrements** :
  - **Insertion d'un enregistrement** : La fonction `inserer_EnregistrementContigue` permet d'insérer un enregistrement dans un fichier existant.

- **Recherche d'un enregistrement** : La fonction `rechercher_EnregistrementContigue` permet de rechercher un enregistrement spécifique dans un fichier.
- **Suppression d'un enregistrement** : La fonction `supprimer_enregistrementContigue` permet de supprimer un enregistrement d'un fichier.

- **Modification des fichiers** :
  - **Tri des enregistrements d'un fichier** Une fonction de tri a été implémentée pour assurer l'organisation interne des enregistrements d'un fichier dans la mémoire secondaire. Cette fonction, prenant comme paramètres la structure `MemoireSecondaire` et le nom du fichier concerné, applique un tri par identifiants (ID) en parcourant tous les blocs associés au fichier. Le tri est effectué en utilisant l'algorithme d'insertion sur chaque bloc individuel, garantissant ainsi que les enregistrements sont ordonnés de manière croissante selon leurs ID. Cette fonctionnalité permet de maintenir une cohérence et une lisibilité des données tout en facilitant les opérations ultérieures de recherche ou de traitement sur les fichiers. Le tri est appelé automatiquement après la création et le remplissage des fichiers.
  - **Renommage d'un fichier** : La fonction `renommer_Fichier` permet de renommer un fichier.
  - **Défragmentation d'un fichier** : La fonction `defragmenter_Fichier` permet de réorganiser les blocs d'un fichier pour optimiser son allocation.
  - **Compactage de la mémoire secondaire:** La fonction `compactage_Memoire_SecondaireC` permet de compacter globalement la mémoire secondaire.
  - **Suppression d'un fichier** : La fonction `supprimer_FichierContigue` permet de supprimer un fichier entier.

- **Affichage et gestion globale de la mémoire** :
  - **Affichage de la mémoire secondaire et des fichiers** : Les fonctions `afficher_Memoire_Secondaire` et `afficher_contenu_du_fichier` permettent d'afficher le contenu de la mémoire secondaire ainsi que les informations détaillées des fichiers.
  - **Vidage de la mémoire secondaire** : La fonction `vider_Memoire_Secondaire` permet de vider entièrement la mémoire secondaire.

En complément, plusieurs fonctions intermédiaires ont été implémentées pour faciliter le traitement des fichiers et des enregistrements. Parmi celles-ci :

- **Calcul et gestion des blocs** :
  - **Calcul du nombre de blocs** : La fonction `calculer_nombre_blocs` permet de calculer le nombre de blocs nécessaires pour un fichier donné.
  - **Mise à jour après compactage** : La fonction `mettreAJourPremierBlocApresCompactage` permet de mettre à jour le premier bloc d'un fichier après compactage.

- **Interaction utilisateur** :

  - **Confirmation des actions** : La fonction `confirmerAction` permet de confirmer les actions avant leur exécution.
  - **Précisions sur les fichiers et enregistrements** : Les fonctions `preciser_fichier`, `preciser_enregistrement` et `choisir_Type_Systeme_Fichier` permettent de gérer les choix interactifs pour préciser les fichiers, enregistrements ou types de système de fichiers.

- **Génération et manipulation des données** :
  - **Génération de données aléatoires** : La fonction `generer_DonneesAleatoires` permet de générer des données aléatoires pour simuler des enregistrements.
  - **Remplissage d'un fichier avec des données simulées** : La fonction `remplir_FichierCor` permet de remplir un fichier avec des données simulées.

Ces fonctions permettent d'explorer les mécanismes fondamentaux des SGF et d'offrir une base solide pour des extensions futures, telles que l'ajout d'autres modes d'allocation ou l'intégration d'une interface graphique.

2.3 Exemple de scénario de test

Cette section présente les étapes du menu et les scénarios de simulation dans le projet de gestion de systèmes de fichiers. Chaque image illustre une étape du processus, avec des menus permettant d'interagir avec la mémoire secondaire, les fichiers et les enregistrements.

2.4 Initialisation et Menu Principal

L'initialisation du programme commence avec le menu principal, où l'utilisateur peut choisir différentes options, telles que l'initialisation de la mémoire secondaire, la création de fichiers ou l'affichage des informations sur la mémoire. Ce menu offre également des options pour accéder aux fonctionnalités avancées du simulateur.

2.5 Initialisation de la Mémoire

Après avoir sélectionné l'option d'initialisation de la mémoire, l'utilisateur définit les paramètres de la mémoire secondaire, tels que la taille, le mode d'allocation (contigu ou chaîné), et d'autres paramètres nécessaires pour commencer à gérer les fichiers et les enregistrements.

2.6 Affichage de l'État de la Mémoire

Une fois la mémoire initialisée, cette étape montre l'état de la mémoire, ainsi que la présentation des fichiers déjà présents dans la mémoire. Elle permet à l'utilisateur de visualiser la répartition des blocs mémoire entre les différents fichiers.

2.7 Gestion de la Mémoire en Mode Chaîné

Cette image montre un exemple de mémoire secondaire gérée en mode chaîné, où chaque bloc de mémoire est lié à un autre, permettant une gestion flexible de l'espace mémoire pour les fichiers et les enregistrements.

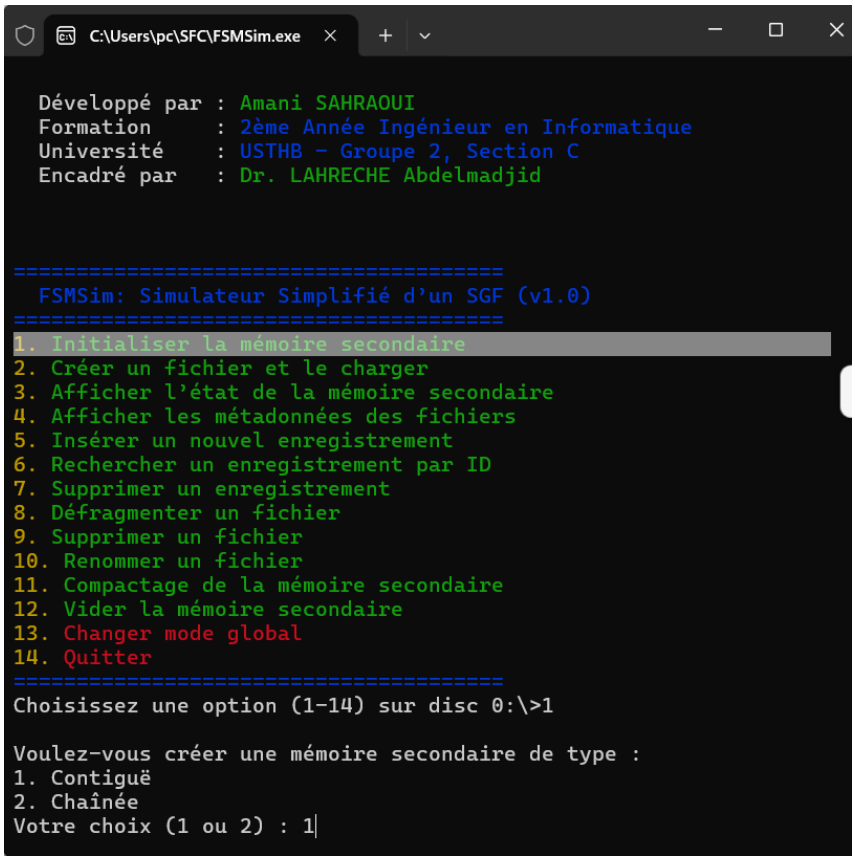

Figure 1: Menu principal et initialisation du programme.

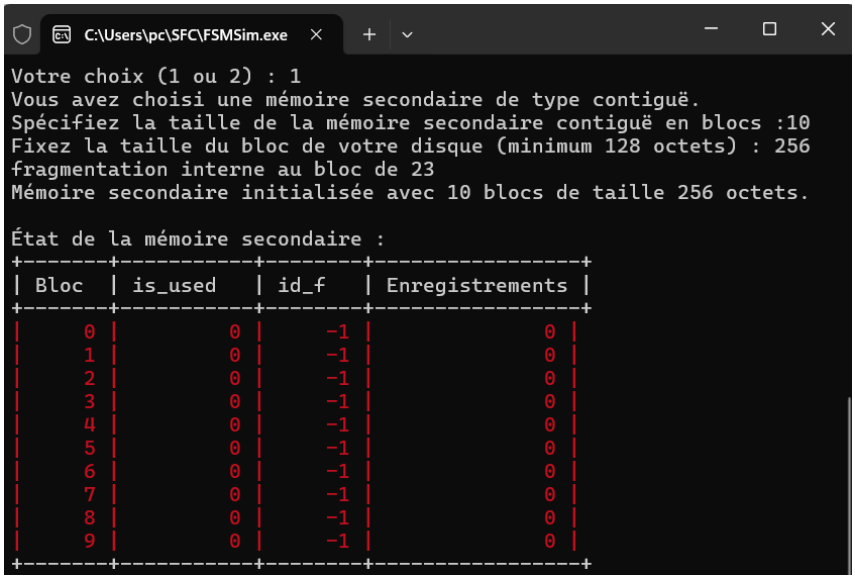

Figure 2: Initialisation de la mémoire secondaire.

2.8 Création d'un Nouveau Fichier

L'utilisateur peut créer un nouveau fichier en spécifiant un nom et une taille. Cette étape montre comment un fichier est ajouté à la mémoire secondaire et comment la mémoire est allouée pour ce fichier.

Cette image montre un fichier récemment créé sans aucun enregistrement à l'intérieur.

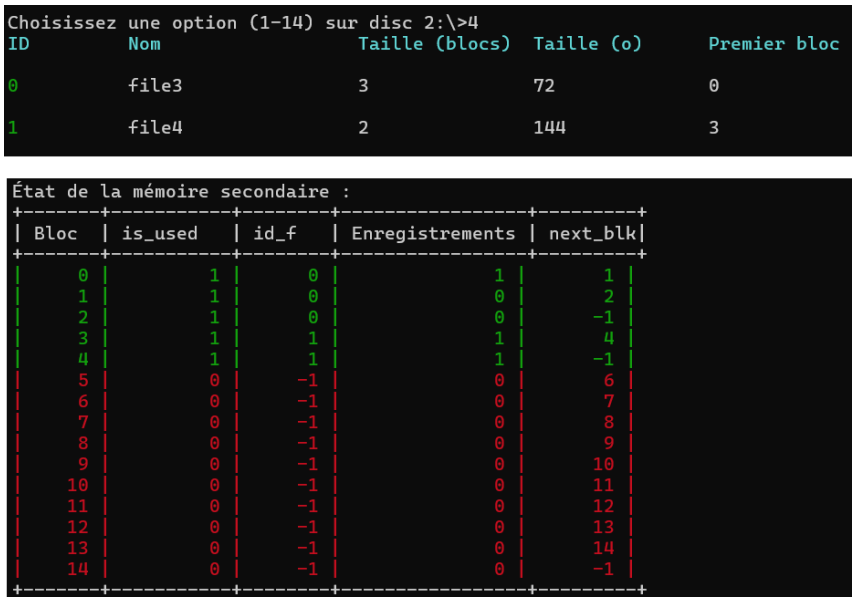

Figure 3: État de la mémoire avec les fichiers présents.

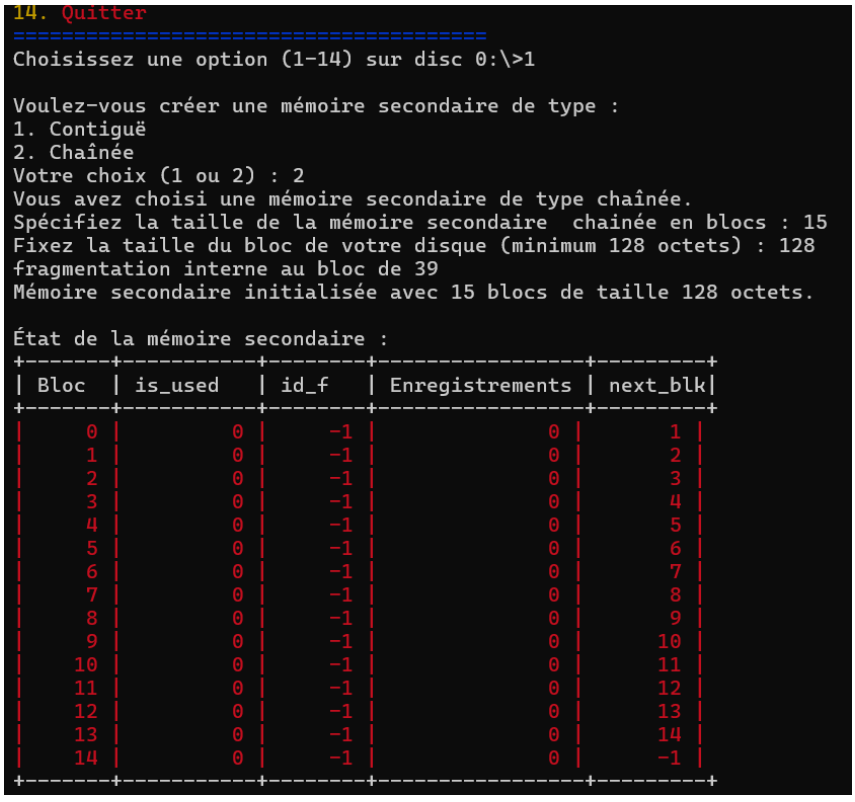

Figure 4: Mémoire secondaire en mode chaîné.

Le fichier est prêt à accueillir des données et des enregistrements dans les étapes suivantes.

2.9 Affichage du Contenu d'un Fichier

Après la création du fichier, l'utilisateur peut afficher son contenu. Cette étape permet de visualiser les enregistrements présents dans le fichier ainsi que leur position dans la mémoire.

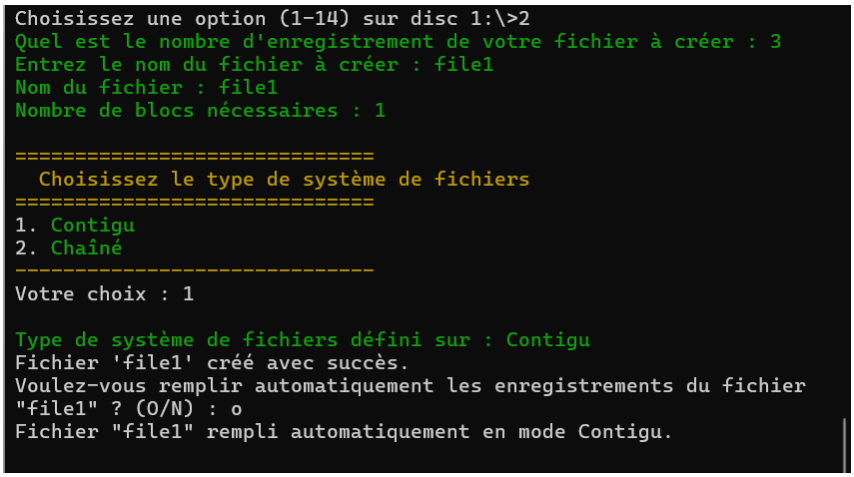

Figure 5: Création d'un nouveau fichier dans la mémoire secondaire.

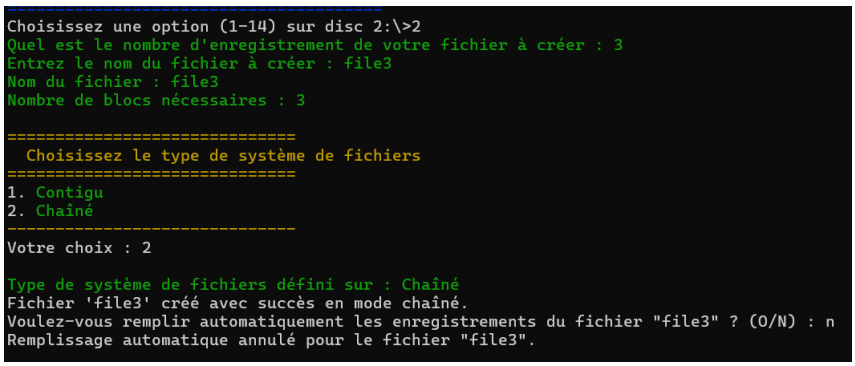

Figure 6: Fichier créé sans enregistrements.

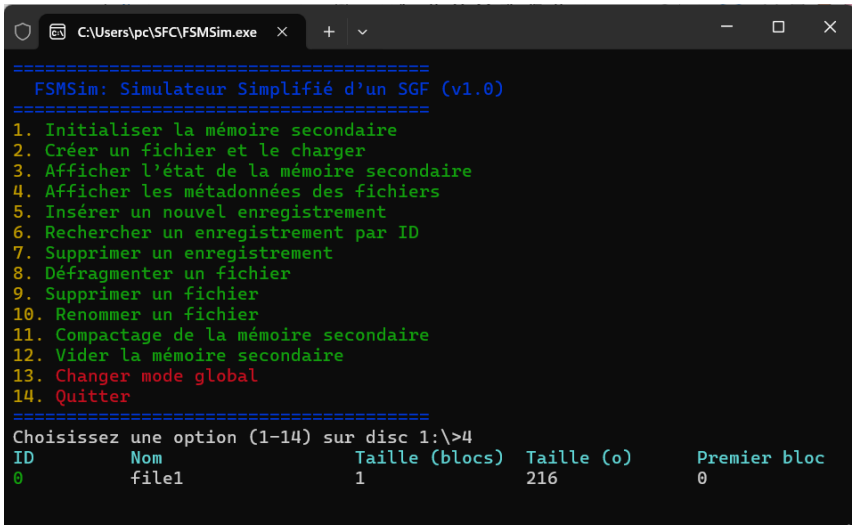

Figure 7: Affichage du contenu d'un fichier.

2.10 Insertion d'un Enregistrement

Cette image montre l'insertion d'un enregistrement dans un fichier. L'utilisateur peut saisir les données à enregistrer et observer comment les enregistrements sont ajoutés au fichier dans la mémoire secondaire.

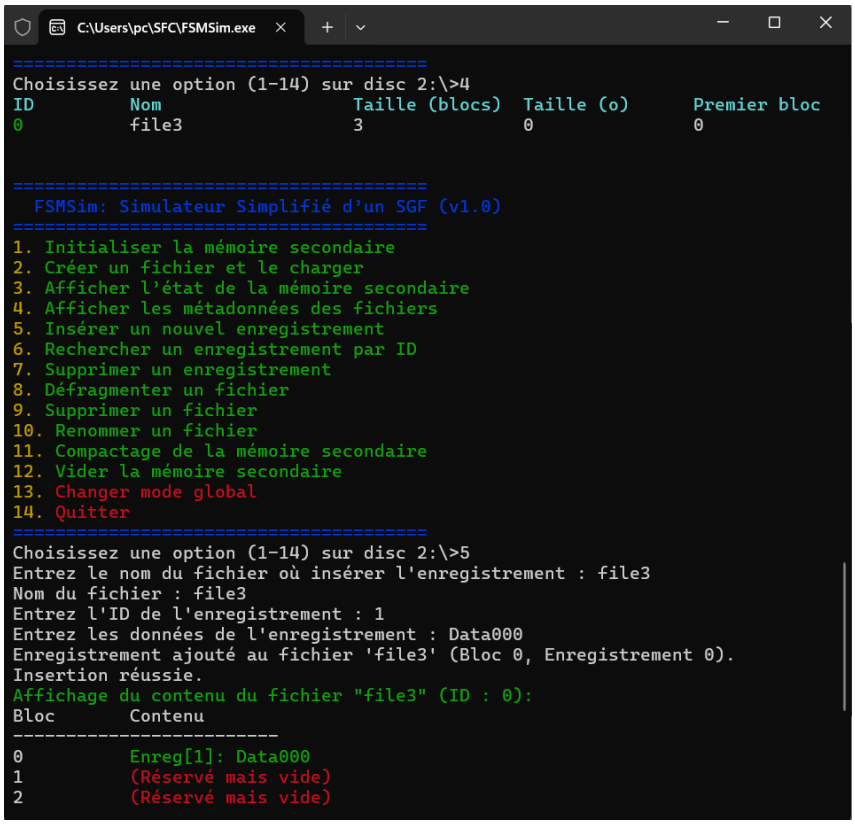

Figure 8: Insertion d'un enregistrement dans un fichier.

Figure 9: Un enregistrement ajouté avec succès.

Après l'ajout d'un enregistrement, l'utilisateur peut vérifier que celui-ci a bien été intégré au fichier, avec la mise à jour de l'état de la mémoire pour refléter l'ajout.

2.11 Amélioration de l'insertion d'un Enregistrement

L'amélioration de l'insertion d'un enregistrement dans un fichier de la mémoire secondaire repose sur l'utilisation de fonctions de tri adaptées au mode de stockage (`mode_interne`). Les fonctions `trier_FichierParIDContigu` et `trier_FichierParIDChaine` permettent de trier les enregistrements par `id`, selon que le fichier soit contigu ou chaîné. Ces fonctions, combinées avec une gestion efficace du `Buffer` (comme l'initialisation, le chargement et l'écriture des blocs), tout en assurant un accès rapide et une gestion cohérente des enregistrements dans la mémoire secondaire. Un exemple est représenté sur la figure 10.

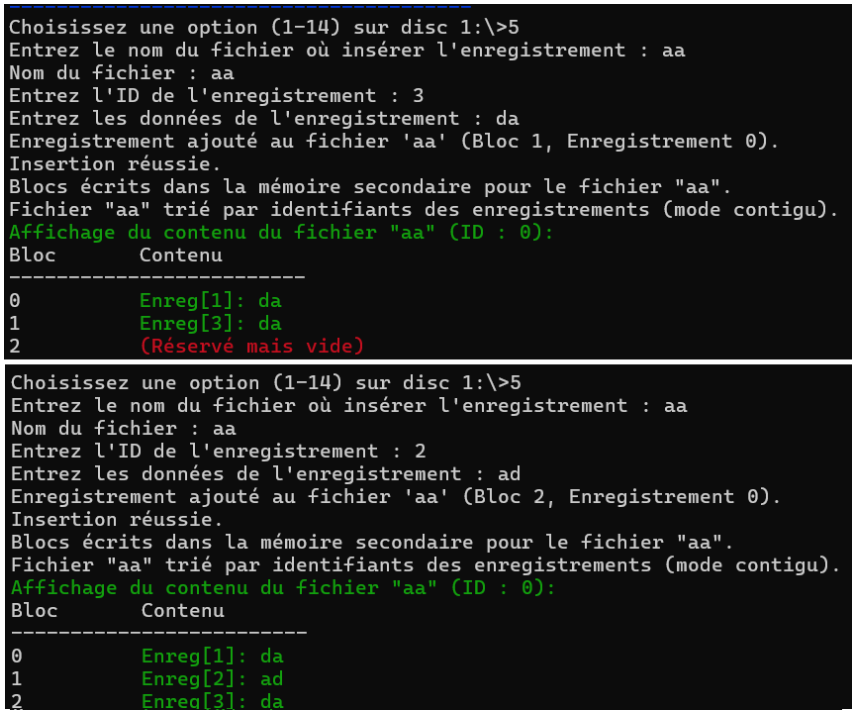

Figure 10: Le tri est effectué avec succès.

2.12 Recherche d'un Enregistrement

Cette image montre le processus de recherche d'un enregistrement dans un fichier. L'utilisateur peut spécifier les critères de recherche, tels que l'ID de l'enregistrement ou d'autres métadonnées, pour localiser l'enregistrement désiré, voir figure 11.

2.13 Suppression d'un Enregistrement

Si nécessaire, l'utilisateur peut supprimer un enregistrement spécifique. Cette étape montre comment les enregistrements peuvent être supprimés de la mémoire et de la structure de données du fichier, voir figure 12.

2.14 Défragmentation d'un Fichier

Pour améliorer l'efficacité de l'allocation mémoire, l'utilisateur peut défragmenter un fichier. Cette étape consiste à réorganiser les blocs de données dans la mémoire pour optimiser l'espace mémoire utilisé, voir 13.

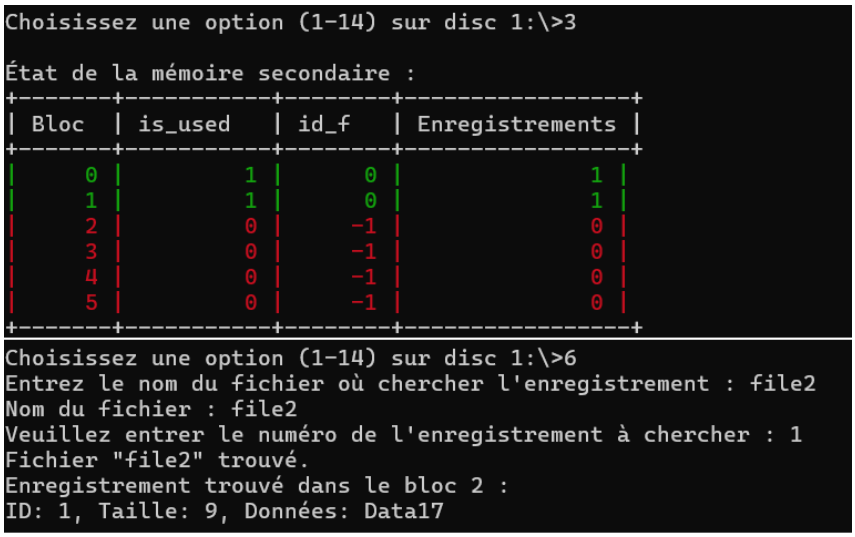

Figure 11: Recherche d'un enregistrement dans un fichier.

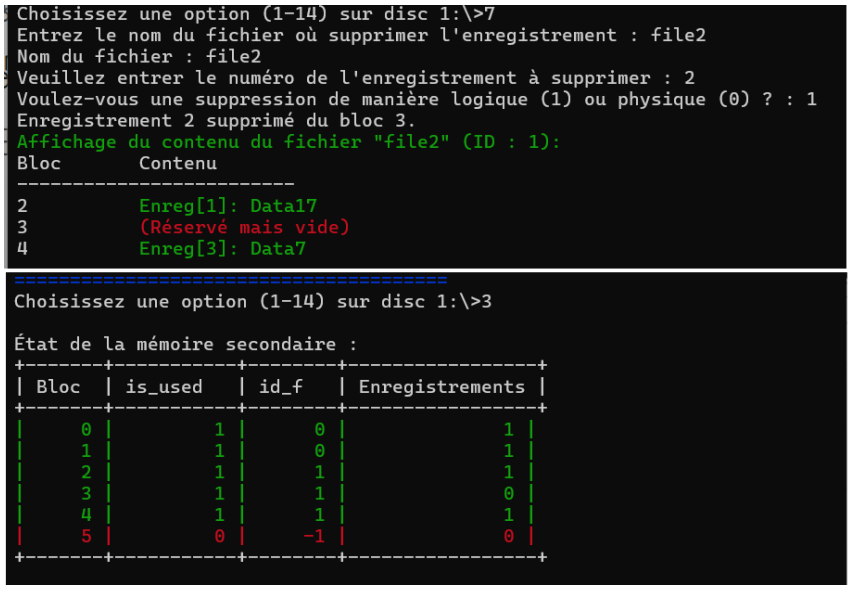

Figure 12: Suppression d'un enregistrement d'un fichier.

2.15 Suppression d'un Fichier

Cette image montre comment un fichier peut être supprimé de la mémoire secondaire. L'utilisateur peut choisir de libérer l'espace mémoire associé au fichier et de le retirer du système, voir figure 14.

2.16 Renommage d'un Fichier

L'utilisateur peut également renommer un fichier existant. Cette fonctionnalité permet de modifier le nom d'un fichier tout en préservant son contenu et sa position dans la mémoire secondaire, voir la figure 15.

Figure 13: Défragmentation d'un fichier pour optimiser l'allocation mémoire.

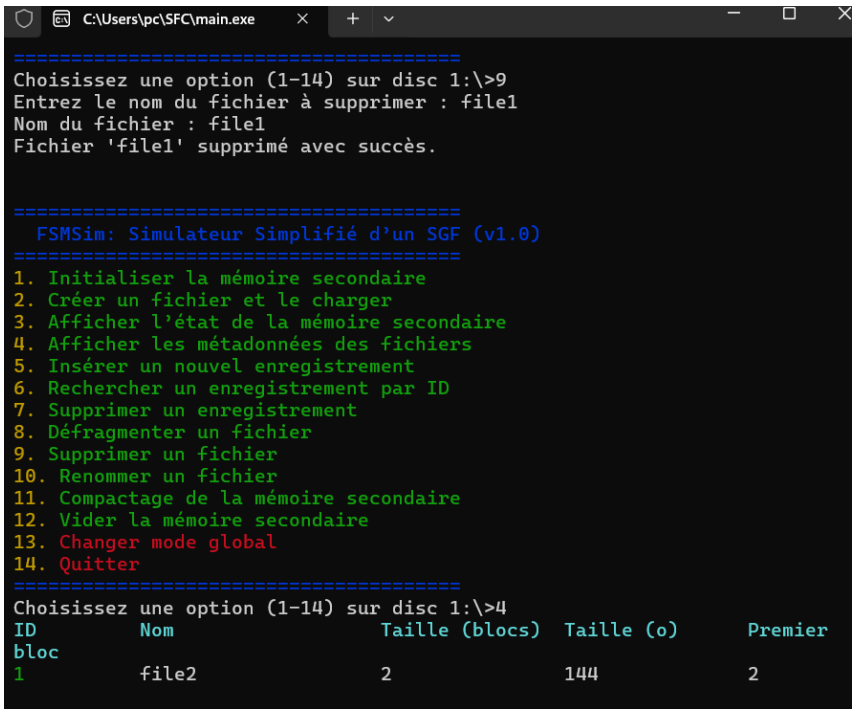

Figure 14: Suppression d'un fichier de la mémoire secondaire.

3 Création, Suppression et Compactage en Mode Chaîné

Dans ce scénario, deux fichiers, `file1` et `file2`, sont créés en mode chaîné, avec une saisie automatique des enregistrements pour chacun d'eux. Après suppression de `file1`, un processus de compactage est exécuté pour réorganiser la mémoire et optimiser l'espace libre, voir figure 16

Ainsi, nous avons exploré l'ensemble des fonctionnalités du menu.

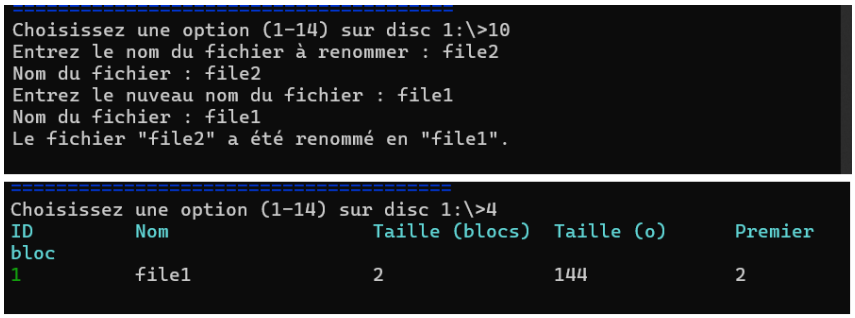

Figure 15: Renommage d'un fichier dans le système.

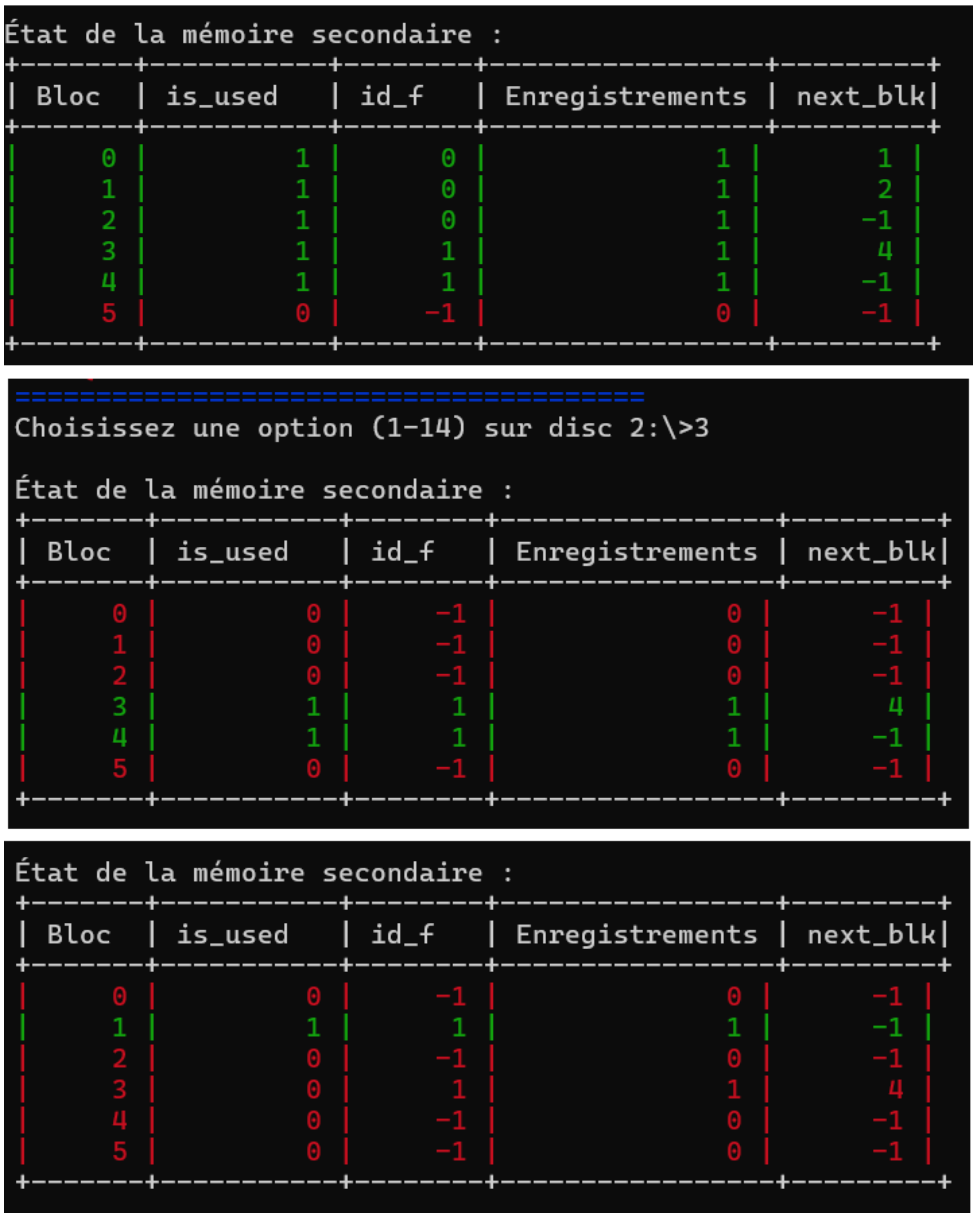

Figure 16: Création, suppression et compactage en mode chaîné.

4 Gestion du Buffer

La gestion du buffer permet d'optimiser les échanges entre la mémoire rapide (RAM) et la mémoire secondaire. Dans ce projet, le buffer agit comme un espace intermédiaire

pour minimiser les accès à la mémoire secondaire en stockant temporairement les blocs nécessaires.

4.1 Fonctionnalités

- **Initialisation :** La fonction `initialiser_buffer` alloue dynamiquement un espace pour le buffer et initialise ses paramètres.
- **Lecture :** `lire_du_buffer` recherche un bloc dans le buffer et le retourne s'il est présent. Sinon, la fonction `lire_memoire` charge le bloc depuis la mémoire secondaire et l'ajoute au buffer.
- **Écriture :** `ecrire_dans_buffer` ajoute un bloc au buffer ou remplace le bloc le plus ancien (stratégie FIFO). Les modifications sont ensuite reflétées dans la mémoire secondaire via `ecrire_memoire`.

Cette partie et belle et bien implémenté dans les fonctions du trie des fichiers lors de l'ajout et la suppression des enregistrements en mode manuel.

5 Difficultés rencontrées

Ayant commencé au début par des programmes simples utilisant des tableaux, j'ai pu comprendre que l'encapsulation permet d'éviter l'utilisation d'un grand nombre de variables globales, une technique facile mais qui n'est pas pérenne à terme. Cependant, l'utilisation des structures, qui encapsulent les variables utiles, a également posé plusieurs difficultés. Par conséquent, le développement du simulateur a présenté plusieurs défis :

- **Gestion de la mémoire dynamique** : La manipulation des pointeurs en C pour allouer et libérer des blocs a nécessité une attention particulière.
- **Configuration de l'environnement** : L'utilisation de VSCode a parfois posé des problèmes de compatibilité avec les bibliothèques standard en C, surtout lorsque j'ai commencé à tester l'environnement graphique SDL.
- **Débogage** : Identifier les erreurs de segmentation et les fuites de mémoire a été complexe en raison de la nature dynamique du projet. Par exemple, basculer entre deux mémoires secondaires, chaînée et contiguë, en même temps (dans le même programme) est une tâche difficile, qui a nécessité beaucoup de gymnastiques.

6 Conclusion et perspectives

Ce projet a permis de mettre en œuvre deux techniques classiques d'allocation dans un système de gestion de fichiers. Les résultats montrent l'importance de choisir la bonne stratégie d'allocation en fonction des besoins.

Perspectives :

- Ajouter un mode hybride combinant les avantages des allocations contiguës et chaînées.
- Intégrer une interface graphique pour une interaction utilisateur plus intuitive.

- Étendre le simulateur pour inclure d'autres modes d'allocation (indexé, multiblindage, etc.).