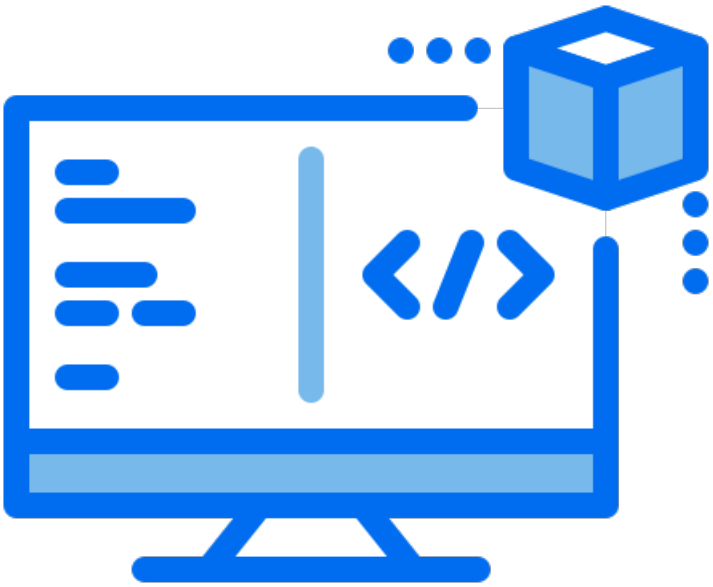
Guide du développeur

pour la bibliothèque de fonctions de gestion de fichiers du projet d'O.S.

par M'GHAIETH Fayçal, KOTOKPO Josué, NASRO Rona.

Mars 2024.



1. La partition

La partition est représentée par un fichier binaire nommé "**myPartition.bin**". Cette partition est définie avec une taille de 32 mégaoctets. Chaque fichier dans la partition est associé à des métadonnées stockées dans une structure nommée **file**.

Chaque bloc de données dans la partition a une taille fixe de 2048 octets. Le nombre total de blocs dans la partition est calculé en divisant la taille de la partition par la taille d'un bloc.

2. Structures et macros

1.1. Variables globales

Plusieurs variables globales sont utilisées pour réaliser la partition :

- **BLOCK_SIZE** : La taille d’un bloc de données, définie à 2048.
- **PARTITION_SIZE :** La taille de la partition, définie sur 32 Mo (32 * 1024 *1024).
- **PARTITION_NAME** : Le nom du fichier représentant la partition, un fichier binaire nommé « myPartition.bin ».
- **nameSize** : La taille du nom d’un fichier dans la partition, en caractères, ici 20 caractères maximum.

1.2. La structure « file »

Cette structure représente les métadonnées d’un fichier. Elle est caractérisée par les éléments suivants :

- **fileName** : un tableau de caractères correspondant au nom du fichier. La taille du tableau, **nameSize**, est de 20 caractères.
- **fileSize** : un entier correspondant à la taille du fichier. 20 par défaut, car un octet est sur 4 bits, avec 5 entiers composants la structure : 5*4 = 20. À cette taille sont ajoutés 40 octets, correspondants à la place laissée au début du bloc pour identifier le fichier concerné, avec ses métadonnées.
- **startBlock** : un entier correspondant au numéro du premier bloc occupé par le fichier dans la partition.
- **isOpen** : un entier qui indique si le fichier est ouvert ou non.
- **isUsed** : un entier qui indique si le bloc comporte déjà un vrai fichier (c’est à dire pas les fichiers placés à l’initialisation de la partition) ou non.
- **cursor** : un entier qui correspond à la position de lecture/écriture dans le fichier.

Ces métadonnées permettent de gérer l'allocation de l'espace sur la partition et de suivre les fichiers et leurs positions dans la partition.

3. Les fonctions de la bibliothèque

3.1. myFormat

La fonction myFormat est responsable de créer et de formater la partition. Elle crée un fichier avec le nom de la partition (PARTITION_NAME) et initialise chaque bloc de données de la partition avec des fichiers vides, c’est à dire des instances de la structure file. Pour réaliser cela, la fonction suit l’algorithme suivant :

1. Elle ouvre le fichier de la partition en utilisant la fonction **open** avec les paramètres suivant : partitionName, O_RDWR | O_CREAT et 0666.
2. Elle initialise une structure file vide, emptyFile, servant à l’initialisation de la partition. C’est à dire que pour chaque bloc de la partition, ces derniers sont remplis avec emptyFile, cela à l’aide de la fonction **write**.
3. Une fois la partition initialisée, celle-ci est fermée avec la fonction **close**.

3.2. myOpen

Le fonction myOpen est responsable d’ouvrir un fichier présent dans la partition à partir du nom de ce fichier, passé en paramètre. Pour réaliser cela, la fonction suit l’algorithme suivant :

1. Elle commence par ouvrir le fichier de la partition spécifié par partitionName en utilisant la fonction open avec le flag O_RDWR pour permettre la lecture et l'écriture dans le fichier et initialise une structure file f utilisée pour récupérer les métadonnées du fichier à ouvrir.
2. Elle parcourt la partition à l’aide de lseek et, pour chaque fichier dans la partition, elle lit ensuite les métadonnées du fichier en utilisant la fonction read et stocke ces métadonnées dans la structure f.
3. Si le champ isUsed est à 1 et que le nom du fichier correspond à celui spécifié par fileName en utilisant strcmp, cela signifie que le fichier recherché a été trouvé dans la partition, donc le booléen indiquant que le fichier est trouvé, nommé found, est mis à 1. Le champ isOpen de la structure f est mis à 1.
4. Si aucun fichier correspondant n'est trouvé dans la partition, c’est à dire que found = 0, la fonction revient au début de la partition en utilisant lseek, puis parcourt à nouveau les fichiers dans les blocs de la partition pour trouver un emplacement disponible pour un nouveau fichier.
5. Lorsqu'elle trouve un bloc disponible, c'est à dire que le champ isUsed est à 0, elle créer le fichier, donc un vrai fichier, ayant le nom du fichier qui était recherché, avec

des métadonnées non nulles comme le champ isOpen = 1, le bloc de départ enregistré, le champ isUsed = 1, et le champ cursor rempli.

6. La partition est ensuite fermée puis un pointeur de type file copiant les métadonnées enregistrées dans f et nommé openedFile est créé puis retourné par la fonction.

3.3. mySeek

La fonction mySeek permet le déplacement du curseur de lecture et d'écriture dans un fichier ouvert en fonction d'un décalage spécifié par l'utilisateur et de la base de référence choisie. Cela permet de positionner le curseur à différents endroits dans le fichier pour lire ou écrire des données à partir de cette position. Cette fonction suit le déroulement suivant :

1. La fonction prend en paramètres un pointeur vers une structure file (f) représentant le fichier dans lequel effectuer le déplacement, une variable offset correspondant au nombre d'octets qu'il faut pour se déplacer, et une variable base qui spécifie la référence à partir de laquelle effectuer le déplacement (début du fichier, position actuelle, ou fin du fichier.
2. Elle commence par vérifier si le pointeur vers la structure file est valide (f!= NULL) et si le fichier est ouvert (f->isOpen == 1). Si ce n'est pas le cas, la fonction ne fait rien et s'arrête.
3. Ensuite, elle calcule la nouvelle position du curseur en fonction du offset et de la base spécifiée :
    - Si la base est SEEK_SET, correspondant au début du fichier, la nouvelle position est définie comme étant le offset + le bloc de départ du fichier + la taille d'un fichier vide.
    - Si la base est SEEK_CUR, correspondant à la position actuelle, la nouvelle position est définie comme étant la position actuelle du curseur dans le fichier (f->cursor) + le offset.
    - Si la base est SEEK_END, correspondant à la fin du fichier, la nouvelle position est définie comme étant la taille du fichier + le offset.
4. Elle vérifie ensuite bien que la nouvelle position calculée n'est pas négative. Si elle est négative, cela signifierait que le déplacement ferait reculer le curseur avant le début du fichier, chose impossible. Pour y remédier, la nouvelle position est fixée à 0.
5. À la fin, elle met à jour la position actuelle du curseur dans la structure f avec la nouvelle position calculée.

3.4. myRead

La fonction myRead permet la lecture des données à partir d'un fichier ouvert dans la partition en utilisant le curseur de fichier pour positionner la lecture, puis en lisant les

données à partir de cette position dans le fichier. Cette fonction est basée sur l'algorithme suivant :

1. La fonction prend en paramètres un pointeur vers une structure file représentant le fichier à partir duquel lire les données (f), un tampon (buffer) où stocker les données lues, et le nombre d'octets à lire (nBytes).
2. Elle commence par vérifier si le pointeur est valide (f == NULL), si le tampon de données est valide (buffer == NULL) ou si le fichier est ouvert (!f->isOpen). Si l'une de ces conditions n'est pas satisfaite, la fonction retourne -1 pour indiquer une erreur de lecture.
3. Ensuite, elle ouvre la partition en mode lecture seule (O_RDONLY) en utilisant la fonction open. Si l'ouverture échoue, elle affiche un message d'erreur à l'aide de perror et retourne -1.
4. Elle utilise lseek pour déplacer le curseur de lecture dans le fichier de la partition à la position indiquée par le curseur de lecture actuel dans la structure file (f->cursor). Cela positionne le curseur de lecture à l'endroit où les données doivent être lues dans le fichier de la partition.
5. Ensuite, elle utilise read pour lire les données à partir du fichier de la partition dans le tampon de données spécifié. Le nombre d'octets réellement lus est stocké dans la variable bytesRead.
6. Elle utilise ensuite une boucle while pour déterminer la taille des données réellement lues en recherchant la présence du caractère nul ('\0') dans le tampon. Cela permet de déterminer la taille exacte des données lues, car read peut lire plus ou moins d'octets que ceux spécifiés par nBytes.
7. Elle ferme ensuite le descripteur de fichier de la partition à l'aide de la fonction close.
8. Enfin, elle retourne le nombre d'octets réellement lus à partir du fichier de la partition. Ce nombre peut être différent de nBytes en fonction de la taille réelle des données lues.

3.5. myWrite

La fonction myWrite est utilisée pour écrire des données dans un fichier ouvert dans la partition. Elle suit le déroulement suivant :

1. D'abord, la fonction vérifie si le pointeur vers la structure file (f) est valide, si le tampon de données (buffer) est valide et si le fichier est ouvert. Si l'une de ces conditions n'est pas remplie, la fonction retourne -1 pour indiquer un échec de l'écriture.
2. Ensuite, elle ouvre le fichier de la partition elle-même en mode lecture/écriture (O_RDWR) en utilisant la fonction open. Si l'ouverture du fichier échoue, elle affiche un message d'erreur à l'aide de perror et retourne -1.

3. Elle utilise lseek pour positionner le curseur de lecture/écriture dans le fichier de la partition à la position indiquée par le curseur de fichier actuel (f->cursor). Cela place le curseur de fichier à l'endroit où les données doivent être écrites.
4. Ensuite, elle utilise write pour écrire les données contenues dans le tampon (buffer) dans le fichier de la partition. Le nombre d'octets réellement écrit est stocké dans la variable bytesWritten.
5. Elle vérifie si une erreur s'est produite lors de l'écriture en vérifiant si write a renvoyé -1. Si c'est le cas, elle affiche un message d'erreur à l'aide de perror, ferme la partition avec close et retourne -1 pour indiquer un échec de l'écriture.
6. Elle utilise lseek à nouveau pour obtenir la position actuelle du curseur dans le fichier de la partition. Cette position indique le nombre d'octets du début de la partition jusqu'à l'emplacement du dernier caractère écrit.
7. Elle met à jour la taille du fichier dans les métadonnées de la structure file (f->fileSize) en calculant la différence entre la position actuelle du curseur et le bloc de départ du fichier.
8. Elle utilise lseek pour positionner le curseur au début du fichier de la partition afin d'écrire les métadonnées mises à jour. Ensuite, elle écrit ces métadonnées dans le fichier de la partition à l'aide de write.
9. Enfin elle met à jour le curseur de fichier (f->cursor) en ajoutant le nombre d'octets réellement écrit, puis elle ferme la partition à l'aide de close et retourne le nombre d'octets réellement écrit (bytesWritten) pour indiquer le succès de l'écriture.

3.5. visualisation

La fonction visualisation permet, comme son nom l'indique, de visualiser la partition avec les parties réservées et libres évoluant en cours de programme. Pour ce faire, elle suit l'algorithme suivant :

1. Elle commence par ouvrir le fichier de la partition en mode lecture seule à l'aide de la fonction open. Si l'ouverture du fichier échoue, elle affiche un message d'erreur à l'aide de perror.
2. Ensuite, elle initialise un compteur pour suivre le nombre de blocs utilisés sur la partition. Ce compteur est initialisé à zéro.
3. La fonction parcourt ensuite chaque bloc de la partition. Pour chaque bloc, elle lit les données correspondantes dans le fichier de la partition et examine la structure de fichier pour déterminer si le bloc est utilisé ou non. Si le bloc est marqué comme utilisé dans la structure de fichier, elle incrémente le compteur de blocs utilisés.
4. Après avoir parcouru tous les blocs de la partition, elle calcule la quantité d'espace utilisé en multipliant le nombre de blocs utilisés par la taille de chaque bloc. À la fin, elle affiche le résultat de la quantité d'espace utilisé sur la partition.

3.6. size

Cette fonction permet d’obtenir la taille d’un fichier. Si le fichier est null, une erreur est affichée, sinon, la fonction retourne la taille du fichier (f->fileSize) moins la taille d’un fichier vide, obtenue avec la fonction sizeOfEmptyFile(), une fonction qui retourne la taille d’un fichier vide (sizeof(f.fileName) + 5 * sizeof(int)).

3.7. A propos des auteurs

La réalisation de cette bibliothèque a été réalisée par KOTOKPO Josué, M’GHAIETH Fayçal et NASRO Rona. Ci-dessous les réalisations de chacun :

KOTOKPO Josué	**M’GHAIETH Fayçal**	**NASRO Rona**
myWrite	myOpen	myFormat
myRead	mySeek	visualisation et size
Tests pour myWrite et myRead	Tests pour myOpen et mySeek	Tests pour myFormat, size et visualisation
main()	Guide développeur	Guide utilisateur