Série d'exercices #15

IFT-2245

9 avril 2019

15.1 Allocation des blocs

1. Expliquer comment fonctionne chacune des 4 méthodes d'allocation suivantes : blocs contigus, blocs chaînés, blocs indexés, *extents*.
2. Contraster les avantages et inconvénients de chacune de ces méthodes.

15.1 Allocation des blocs (Solution)

Blocs contigus

Alloue simplement les fichiers comme une séquence de blocs contigus.
Utile si on a une mémoire limité et que les fichiers changent peu (idéalement pas du tout).

15.1 Allocation des blocs (Solution)

Blocs contigus

Alloue simplement les fichiers comme une séquence de blocs contigus. Utile si on a une mémoire limité et que les fichiers changent peu (idéalement pas du tout).

Avantages

- Simple
- Peu de méta-donnée
- Accès séquentielle efficace.

Inconvénients

- Fragmentation externe
- Rigide (taille de fichier)

15.1 Allocation des blocs (Solution)

Blocs chaînés

Chaque bloc contient un pointeur vers le bloc suivant.

15.1 Allocation des blocs (Solution)

Blocs chaînés

Chaque bloc contient un pointeur vers le bloc suivant.

Avantages

- Règle la fragmentation externe.
- Flexible (taille de fichier)

Inconvénients

- Utilisation mémoire des pointeurs (solution : cluster avec un pointer par cluster).
- Accès séquentiel coûteux lorsque les blocs ne sont pas contigus.
- Fragile (La perte d'un pointeur de liste peut entrainer un perte de beaucoup de données).

15.1 Allocation des blocs (Solution)

Blocs indexés

Les pointeurs sont conservés dans un bloc d'indices.

15.1 Allocation des blocs (Solution)

Blocs indexés

Les pointeurs sont conservés dans un bloc d'indices.

Avantages

- Règle les problèmes de fragmentation externe
- Permet l'accès aléatoire.

Inconvénients

- Taille du bloc à allouer (solution : hiérarchie).
- Accés séquentiel pas très efficace si la table n'est pas en mémoire et que les blocs ne sont pas contigus.

15.1 Allocation des blocs (Solution)

Extents

Modification au système d'allocation contigus où il est possible d'allouer une nouvelle zone mémoire contigue.

15.1 Allocation des blocs (Solution)

Extents

Modification au système d'allocation contigus où il est possible d'allouer une nouvelle zone mémoire contigue.

Avanatges

- Bénéfice de l'allocation contigue.
- Flexibilité pour les changements de taille.

Inconvénients

- Fragmentation interne selon la taille des extents.
- Fragmentation externe si extents de taille variable.

15.2 Systèmes de fichiers

Soit un système de fichiers de style “Unix” tel que le FFS de BSD ou ext2 de Linux, avec des blocs indexés.

1. Lister tous les blocs qu'il faut modifier sur le disque lors de l'exécution de la fonction `open` qui crée un fichier (de taille zéro).
2. Spécifier l'ordre dans lequel ces opérations devraient être exécutées pour minimiser l'impact potentiel d'un crash à mi-course.
3. Sur la base de l'ordre précédent, indiquer après chaque opération quels problèmes apparaîtraient en cas de crash à ce moment.

15.2 Systèmes de fichiers (Solution)

- File control block (fcb ou inode sous linux).
- Directory entry.
- Free list
- superblock s'il y a des métadonnées à modifier (i.e. taille totale allouées, dernière modification, etc).

15.2 Systèmes de fichiers (Solution)

1. Création du fcb :
   - avec taille 0
   - pointeur vers le bloc d'indice à 0 (0 est une adessse généralement réservée sur un système de fichier).
2. Modification de la *free list*
3. Ajout du fichier dans l'entré de répertoire
   - Ajout du nom de fichier
   - Puis modification du pointeur

15.2 Systèmes de fichiers (Solution)

1. Création du fcb :
   S'il y a un crash ici, le fichier sera considéré comme non-crée suite au crash.
2. Modification de la *free list*
   S'il y a un crash on perd 1 bloc (Qui sera récupéré lors du prochain `fsck`)
3. Ajout du fichier dans l'entré de répertoire :
   S'il y a un crash suite à l'ajout du nom, on peut facilement constater la présence d'un pointeur vers 0 et considérer que le fichier n'a pas été crée.

Le block d'indice n'a pas nécessairement à être crée puisque le fichier est vide.

15.3 Mount multiples (11.2)

Quels problèmes peuvent apparaître quand on autorise un système de fichiers à être *monté* à plusieurs endroits en même temps.

15.3 Mount multiples (11.2)

Quels problèmes peuvent apparaître quand on autorise un système de fichiers à être *monté* à plusieurs endroits en même temps.

Solution

Les problèmes sont principalement dû à la confusion que peut entrainer ce genre de fonctionnalité.

Par exemple, il peut ne pas être transparent pour l'utilisateur que deux fichiers référencés par deux points de montages différents soit en fait le même fichier.

Un utilisateur pourrait alors supprimer le fichier en pensant qu'il possède une copie.

15.4 VFS (11.8)

Discuter de l'usage d'une abstraction nommée *VFS* pour permettre à un système d'exploitation d'utiliser facilement plusieurs sortes de systèmes de fichiers.

15.4 VFS (11.8)

Discuter de l'usage d'une abstraction nommée *VFS* pour permettre à un système d'exploitation d'utiliser facilement plusieurs sortes de systèmes de fichiers.

Solution

Discuté

15.5 Blocs libres (11.11)

Soit un système où l'espace libre est maintenu dans une liste chaînée de blocs libres.

1. Supposons que le pointeur sur le premier bloc libre est perdu. Le système peut-il reconstruire la liste des blocs libres ?

15.5 Blocs libres (11.11)

Soit un système où l'espace libre est maintenu dans une liste chaînée de blocs libres.

1. Supposons que le pointeur sur le premier bloc libre est perdu. Le système peut-il reconstruire la liste des blocs libres ?

Solution

1. Oui, il suffit de parcourir la hiérarchie de fichier et de marquer les blocs utilisés. Les blocs libres ne seront pas ajouté à cette liste.

15.6 Optimisation et pannes (11.13)

Discuter comment les optimisations de performance pour les systèmes de fichiers peuvent introduire des problèmes de cohérence en cas de panne.

15.6 Optimisation et pannes (11.13)

Discuter comment les optimisations de performance pour les systèmes de fichiers peuvent introduire des problèmes de cohérence en cas de panne.

Solution

Une optimisation possible est de retarder une écriture le plus longtemps possible en espérant qu'une future écriture écrase celle initiale. Une telle stratégie est problématique puisqu'elle ne considère pas la possibilité d'une panne qui serait désastreuse pour l'intégrité des fichiers et du système de fichiers.

15.7 Indexage indirect progressif (11.15)

Soit un système de fichiers de type “Unix File System” avec des blocs de 8KB, où un pointeur sur un bloc occupe 4 bytes, et où chaque inode contient 12 pointeurs sur des blocs directs, 1 pointeur sur un bloc indirect, 1 pointeur sur un bloc doublement indirect, et 1 pointeur sur un bloc triplement indirect.

- Quelle est la taille maximum d'un fichier ?
- Combien d'accès disques sont nécessaires (en présumant que le cache est vide) pour accéder au contenu d'un petit fichier dans /a/b/c ?

15.7 Indexage indirect progressif (11.15)

Soit un système de fichiers de type “Unix File System” avec des blocs de 8KB, où un pointeur sur un bloc occupe 4 bytes, et où chaque inode contient 12 pointeurs sur des blocs directs, 1 pointeur sur un bloc indirect, 1 pointeur sur un bloc doublement indirect, et 1 pointeur sur un bloc triplement indirect.

- Quelle est la taille maximum d'un fichier ?
- Combien d'accès disques sont nécessaires (en présumant que le cache est vide) pour accéder au contenu d'un petit fichier dans `/a/b/c` ?

Solution

- $8\text{KB} * (12 + 2048 + 2048^2 + 2048^3) = 64\text{GB}$
- 10 opérations sont nécessaires. Une lecture pour obtenir l'inode de chacun des dossiers (`/`, `/a/`, `/a/b/` et `/a/b/c/`), une lecture pour obtenir (on présume) l'un des 12 premiers blocs pour lire le contenu des dossiers. Une lecture pour obtenir l'inode du fichier et une lecture pour accéder à l'un des blocs.

15.8 11.17

Comparer l'usage d'un *RAM disque* par rapport à l'usage de la même mémoire comme un cache.