Partie 8 : Systèmes de fichiers

Le corrigé

Solution 1 :

a) Chaque bloc de données peut contenir 256 pointeurs (numéros de blocs). Il faut :
- (12 * 1k) pour direct,
- (256 * 1K) pour ind. simple,
- (256 *256 * 1K) pour ind. double,
- (256 * 256 * 256 * 1K) pour ind. Triple

Taille = (12 * 1K) + (256 * 1K) + (256 * 256 * 1K) + (256 * 256 *256 * 1K) = (12 + 256 + 256*256 + 256*256*256) K = 16,843,020 K octets.

b) Si on considère un fichier contenant 100,000 octets, on voit que :
100,000 = 97 * 1024 + 672. Il faudra donc 98 blocs pour conserver les données de ce fichier.
L*'i-noeud* ne dispose que de 12 pointeurs directs, il va donc falloir utiliser des blocs supplémentaires (98-12=86) pour conserver le reste des données. Comme 86<256, ces blocs de données seront accessibles à partir du pointeur indirect simple du *i-noeu*d.
Une partie d'un bloc sera nécessaire pour conserver les 86 pointeurs sur des blocs de données.
Le nombre total de blocs utilisés est donc 98+1 = 99.

Solution 2 :

Notre fichier compte 20,000,000 = 4882 * 4096 + 3328 octets. Il faudra donc 4883 blocs pour conserver les données de ce fichier. Il faudra aussi leur ajouter des blocs qui vont être utilisés pour stocker des pointeurs vers ces blocs de données. On effectue donc le calcul suivant (sur les blocs) :

– Le fichier compte 4883 blocs de données.

– Les pointeurs directs de l*'i-noeud* permettent d'accéder à 12 de ces blocs. Il reste donc 4871 blocs de données pour lesquels l'accès se fera à travers l'un des liens indirects.

– Le pointeur de lien indirect simple pointe sur un bloc qui contient 1024 numéros de blocs (pointeurs vers des blocs de données). Nous avons donc ajouté 1 bloc de pointeurs, et il reste 4871 - 1024 = 3847 blocs à traiter.

– Le pointeur de lien indirect double permet d'accéder à 1024*1024 blocs, ce qui est plus que suffisant. Il suffit d'utiliser 4 blocs de données pour stocker les 3847 pointeurs de blocs permettant d'accéder aux données restantes.

Fragmentation interne

L'espace alloué mais non utilisé :

– 4 octets sur l'i-noeud (indirection triple non utilisé)

– (1024-4)*4 = 4080 octets dans le bloc sur lequel pointe le pointeur indirect double.

– (4096-3847)*4 = 996 octets dans le dernier bloc de pointeurs alloué.

– Finalement, 768 octets dans le dernier bloc de données.
La fragmentation interne totale sur le disque est donc de 5848 octets.

Solution 3 :

1) Lorsqu'un processus demande l'ouverture d'un fichier ordinaire existant, le système de fichier :
   - récupère en analysant le chemin d'accès l'i-nœud du fichier,
   - accède à la table des i-nœuds, récupère toutes les informations concernant le fichier, vérifie si l'ouverture est permise. En cas de problème, il retourne – 1 au processus demandeur,
   - En cas de succès, il crée une entrée dans la table des fichiers ouverts pour y insérer des informations sur le fichier notamment son i-nœud, le mode d'ouverture, le pointeur de fichier…. Il crée également une entrée dans la table des descripteurs de fichier du processus demandeur. Cette entrée pointe sur l'entrée créée dans la table des fichiers ouverts. Le processus demandeur reçoit, dans ce cas, la position de l'entrée créée dans la table des descripteurs.

2) a) Dans ce cas, les deux processus partagent le même pointeur de fichier :
Le fichier a été ouvert avant l'appel système fork. L'appel système fork duplique la table des descripteurs de fichier.

   b) Dans ce cas, chaque processus a son propre pointeur de fichier car il y a eu deux demandes d'ouverture de fichier donc création de deux entrées dans la table des fichiers ouverts une pour chaque ouverture de fichier (2 pointeurs de fichier).

3) a) Un accès pour récupérer le répertoire racine (1 bloc)
Un accès pour récupérer le répertoire usr (1 bloc)
Un accès pour récupérer le répertoire cours (1 bloc)
Un accès pour récupérer le répertoire INF3600 (1 bloc)
Un accès pour récupérer le répertoire tps (1 bloc)
L'i-nœud du fichier tp.pdf est dans le répertoire tps.
5 accès.
b) Rapidité d'accès au fichier et Pas de risque de référencer un fichier qui n'existe pas.

Solution 4 :

1) (a)
- Chaque bloc d'index pointe vers 2047 blocs de données soit (8 KO / 4 O)-1.
- Le nombre maximal de numéros de blocs est 2^{32}
- Si m est le nombre maximal de blocs d'index et tous les blocs sont alloués au fichier (données +index), on a :

2^{32} = m + (2047 * m) => m = 2^{32} / 2048 => m= 2^{21}
Le nombre de blocs de données est : 2047 * 2^{21} .

La taille maximale d'un fichier serait de 2 32 blocs dont 2 21 blocs d'index et 2047 *2^{21} blocs de données.
(b)
Le bloc de données j=9000 se trouve dans le bloc d'index i= [9000/K].
Le bloc de données j se trouve dans le bloc de données k=9000%K à l'intérieur du bloc d'index i.
Le nombre de blocs à lire est i+1.
2) (a) Les systèmes d'exploitation de la famille Unix permettent à un même fichier de posséder un nom différent en divers répertoires. Seul le graphe acyclique permet d'avoir cette fonctionnalité.
(b) (i) Le nom d'un fichier n'est pas conservé dans l'i-noeud (inode) car cela ne permettrait pas à un fichier d'avoir des alias (des liens).
(ii) Les noms des fichiers sont conservés dans les entrées des répertoires avec aussi les numéros des inodes. Ainsi, un inode peut être référencé en divers répertoires par des noms différents.
(iii) Pour éviter des cycles dans la structure de répertoire.
3) Avec le polling, le pilote du périphérique doit constamment surveiller la complétion de l'opération E/S. Avec les interruptions, le périphérique envoie un signal au pilote lorsque l'opération est complétée et ceci permet d'éviter l'attente active du pilote.

Solution 5 :

a) Un bloc peut contenir au maximum 512 adresses. La taille maximale est de 1034 blocs (11 + 511 + 512). Ceci correspond à 1034 Koctets, donc environ 1Moctets.
b) Il s'agit du premier octet du bloc 205. Le pointeur à retrouver est donc à la poistion 194 du premier bloc d'indirection simple. Le bloc désiré est donc à la position 0xB200 sur le disque dur.

Solution 6 :

1- Récupérer le nom de fichier du chemin d'accès path1;
Concaténer le nom au path2 ;
link (path1, path2);
unlink (path1);
2- non, car, d'une part, les liens physiques ne sont pas permis entre fichiers gérés par des systèmes de fichiers différents et d'autre part, les liens symboliques ne sont pas appropriés car la suppression de path1 peut entraîner l'élimination du fichier physique.

Solution 7 :

a) Le nombre de clusters est obtenu en divisant la taille de la partition par la taille d'un cluster. On obtient 524288 clusters. Comme il faut 1 bit par cluster, cela conduit à une taille du fichier bitmap de 64 Ko.
b) L'entête indique les numéros de blocs logiques dont les numéros physiques correspondants sont dans l'enregistrement. Il sert à accélérer la localisation des blocs physiques d'un fichier.