TP 2 – Travail sur les liste chainée.

Q. Giorgi.

Recommandations

Lisez très attentivement le sujet

ATTENTION : La présentation, la lisibilité, ainsi que le **respect des consignes** seront prises en compte dans la notation.

Comme pour chaque TP, vous :

- Réaliserez un makefile pour la compilation des fichiers sources.
- Utiliserez les options de compilation -Wall -g
- Devrez rendre, selon les consignes de l'enseignant, un code source testé et qui se compile correctement !

Travail préparatoire, rappels et compléments de cours.

Lecture préparatoire :

- Lisez (et comprenez !) le document "Collections.pdf" fourni avant le TP.

- Lisez cet article afin de connaitre le format d'un fichier FAT :

https://fr.wikipedia.org/wiki/File_Allocation_Table

- Exercice d'entrainement à réaliser avant le TP (exercice non noté, des indications seront données sur Chamilo pour ceux qui n'y arriveraient pas, il est très fortement recommandé de vous tester sur ces exercices qui normalement doivent vous prendre pas plus de 30 ou 40 minutes)

A partir du fichier exoListeChaine.c, completer les procédures dont les protoypes de fonctions sont :

void insertionTete(Lnode **ph,char item) ;

void insertionQueue(Lnode **ph,char item) ;

void suppressionTete(Lnode **ph) ;

void suppressionQueue(Lnode **ph) ;

Travail préparatoire à rendre en début de TP (ce travail sera évalué) :

- Pour chaque fonction du fichier fat.h, vous écrirez en français, un commentaire indiquant l'algorithme utilisé pour réaliser la fonction. (pas plus de 10 lignes par fonction)

Problème de gestion de données type FAT

Introduction

Il vous est demandé dans ce problème d'implémenter une solution permettant de gêrer des objets dans un volume de stockage. Ce problème peut s'apparenter à un problème de gestion d'un système de fichiers. Le volume de données étant un disque et les objets (des fichiers).

Afin de faciliter la gestion de ce volume de données, on le découpe en blocs, il est ainsi divisé en 1024 blocs de 512 octets chacuns,

suivant le schéma ci-dessous :

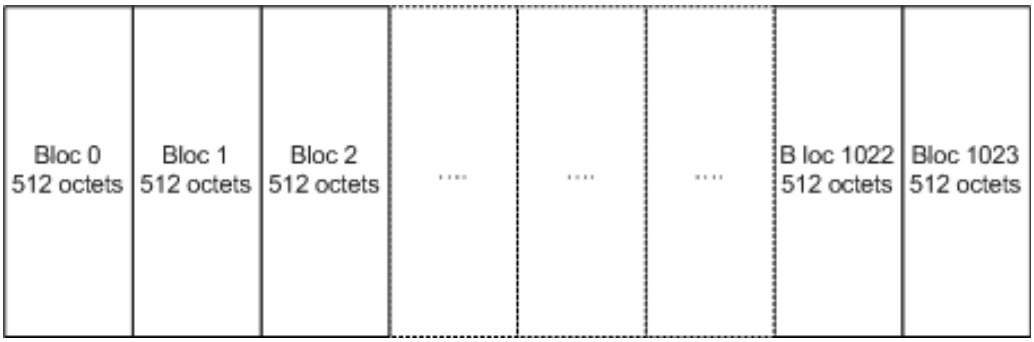

Fig.Schéma du volume de données

Un objet (fichier) est un ensemble de données de taille variable devant être stocké dans ce volume de données.

Un objet, selon sa taille, peut donc utiliser plusieurs blocs de données (pas forcémentt contigüs).

Si la taille de l'objet n'est pas multiple de 512 octets, l'espace restant dans le dernier bloc de l'objet est perdu et n'est pas réutilisé.

On distinguera 2 types d'informations :

- Les données contenues dans les objets (les 1024 blocs de données ci-dessus)

- Les métadonnées qui permettent de décrire les objets (voir la liste chainée et le tableau d'entiers ci-dessous)

Liste chainée:

Une liste chaînée permet ici de définir les objets présents dans le volume de données, cette liste chainée peut être considérée comme le « répertoire » des objets (dans le cas réel, les répertoires sont stockés dans les volumes de données, mais pour les besoins du TP nous le considérons externe). Cette liste chaînée est constituée par la structure contenant les champs suivants :

- Nom (on considère le nom unique)
- Taille
- Propriétaire (noté auteur) codé sur 16 bits

- Numéro du premier bloc de données utilisé (on verra ci dessous comment il est utilisé)
- Pointeur vers l'objet suivant

Cette liste décrit donc les objets qu'il y a dans le volume.

Tableaux d'entiers :

Comme nous venons de le mentionner, un objet peut utiliser plusieurs blocs, pour savoir exactement la liste et l'ordre des blocs utilisés par un objet (qui ne sont pas toujours contigüs), on introduit un tableau d'entiers (de 16bits) non signés [noté FAT pour File Allocation Table], de 1024 éléments. Ce tableau sert aussi à indiquer si les blocs de données sont utilisés ou non.

L'indice du tableau correspond au numéro du bloc dans le volume.

Chacun des éléments de ce tableau contient :

- La valeur 0xFFFF si le bloc correspondant dans le volume est libre
- Si le bloc correspondant dans le volume est utilisé par un objet :
  - l'indice du prochain bloc de données utilisé par un objet.
  - OxFFFE si ce bloc est le dernier bloc de l'objet

Exemple :

Soit l'exemple suivant :

Dans cet exemple nous considérons deux fichiers nommés 'fichier1' et 'fichier2'.

- Le premier fichier est composé de 4 blocs de données (0,1,2,6)
- Le deuxième fichier est composé de 2 blocs de données (5 et 7)

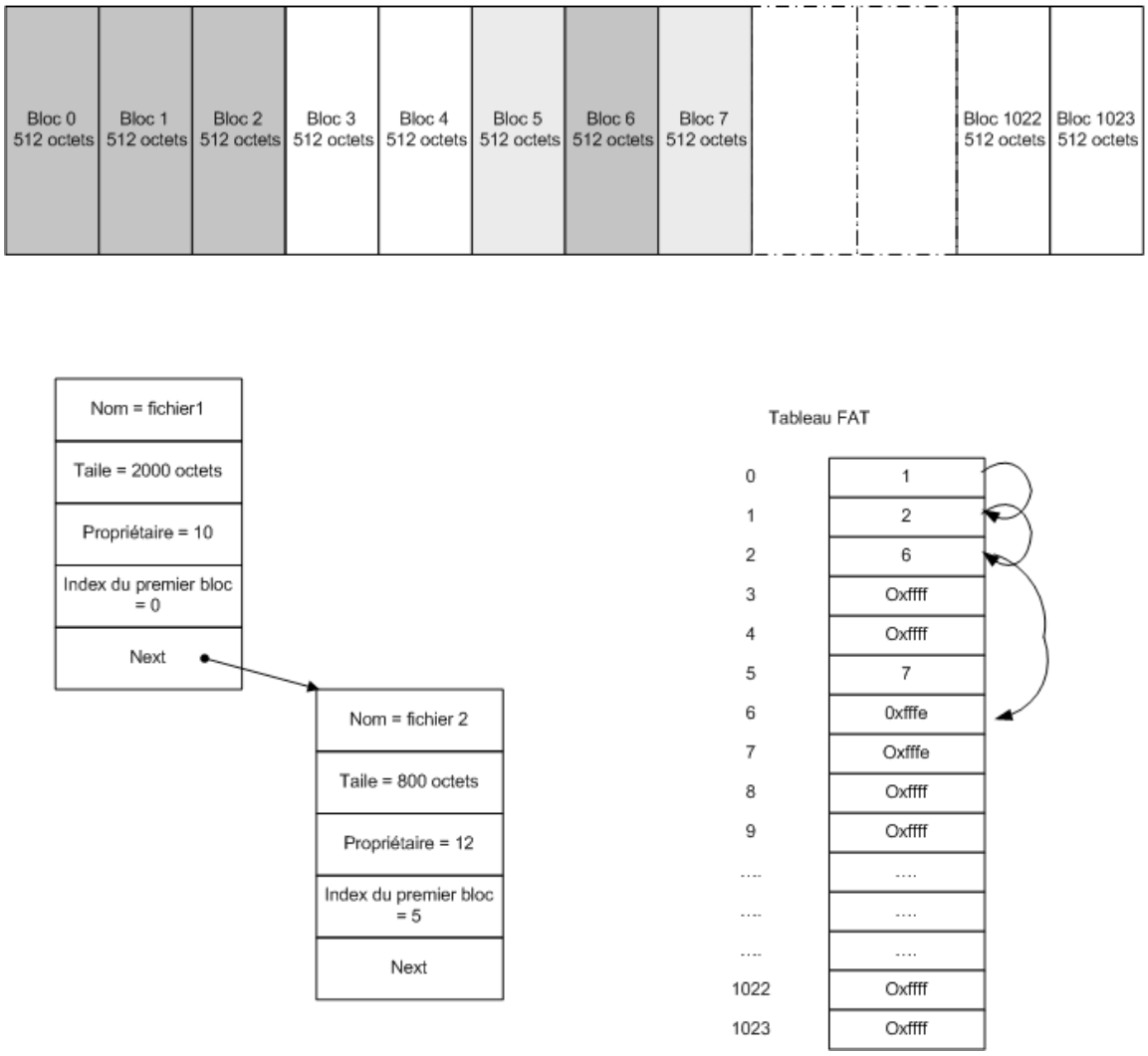

Fig : exemple.

On retrouve à gauche la liste chainée comportant 2 objets.

- le fichier1, de taille 2000 octets (donc 4 blocs), de propriétaire 10, et son premier bloc de données est le bloc 0
- le fichier2, de taille 800 octets (donc 21 blocs), de propriétaire 12, et son premier bloc de données est le bloc 5)

En effet le premier bloc du fchier1 est le bloc 0. (champ index de la structure objet corresponsdante)

or FAT[0]=1, donc le bloc suivant est 1.

or FAT[1]=2, donc le bloc suivant est 2.

or FAT[2]=6, donc le bloc suivant est 6.

or FAT[6]=0xFFFE, donc c'est le dernier bloc de l'objet.

Le premier bloc du fichier2 est le bloc 5. (champ index de la structure objet corresponsdante)

or FAT[5]=7, donc le bloc suivant est 7.

or FAT[7]=0xFFE, donc c'est le dernier bloc de l'objet.

Puis

FAT[3]=0xFFFF, donc ce bloc est libre.

FAT[4]=0xFFFF, donc ce bloc est libre. Etc...

Travail demandé :

Lisez le fichier fat.h

Il vous est demandé de construire les procédures ou fonctions correspondantes aux prototypes des fonctions du fichier "fat.h"

Note : certaines variables pourront être globales.