Chapitre 1

Généralités
- Structures de Fichiers -

Plan du chapitre

1) Les mémoires
Mémoire Centrale, Mémoire Secondaire, Disque Dur, Flash Disk SSD ...

2) Notions de base
Fichier, enregistrement, bloc, buffer, bloc d'entête, E/S physique, gestion de la zone tampon …

3) Manipulation de fichiers en langage C

4) Machine abstraite pour les structures de fichiers

Les différents types de mémoire

Dans l'architecture des ordinateurs actuels, il existe différents niveaux de mémoires :

1- Les registres du processeur
très rapide (qlq ns), **volatile**, taille de quelques mots mémoires

2- La mémoire cache entre processeur et mémoire centrale (MC)
plus rapide que la **MC** (qlq dizaines de ns), **volatile**, très petite taille

3- La mémoire centrale (MC)
rapide (qlq centaines de ns), **volatile**, relativement de petite taille

4- Une zone tampon (jouant le rôle de cache) entre MC et MS
c'est une petite partie de la mémoire centrale réservée par le système ⇒ 'buffer cache'

5- Les différents types de mémoires secondaires (MS)
principalement : disques magnétiques (ou disque durs HDD) et flash-disk (SSD...)
plus lents que **MC**, **non volatiles**, grande taille et moins coûteux que **MC**

6- Les mémoires d'archivage (tertiaire)
Généralement des bandes magnétiques (ou plus rarement des disques optiques)
Très lentes, **non volatile**, grande taille et généralement à très faible coût

Les différents types de mémoire

Dans l'architecture des ordinateurs actuels, il existe différents niveaux de mémoires :

1- Les registres du processeur
très rapide (qlq ns), **volatile**, taille de quelques mots mémoires

2- La mémoire cache entre processeur et mémoire centrale (MC)
plus rapide que la **MC** (qlq dizaines de ns), **volatile**, très petite taille

3- La mémoire centrale (MC)
rapide (qlq centaines de ns), **volatile**, relativement de petite taille

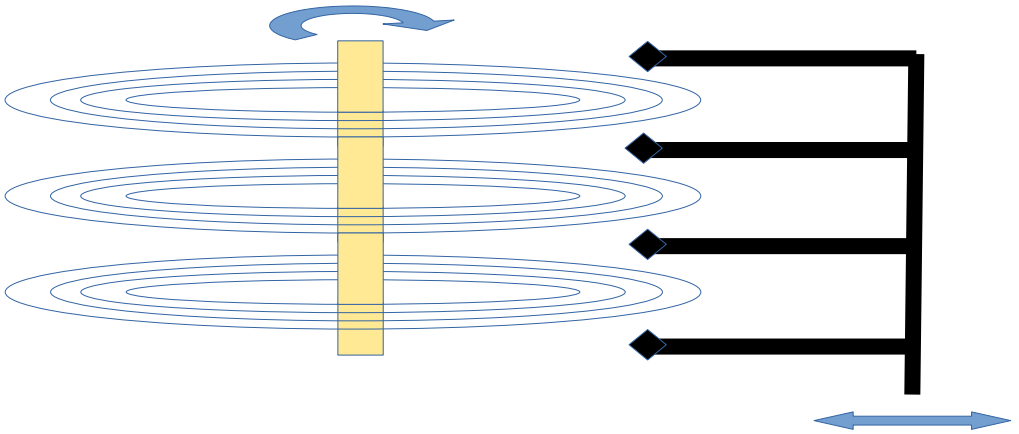
4- Une zone tampon (jouant le rôle de cache) entre MC et MS
c'est une petite partie de la mémoire centrale réservée par le système ⇒ 'buffer cache'

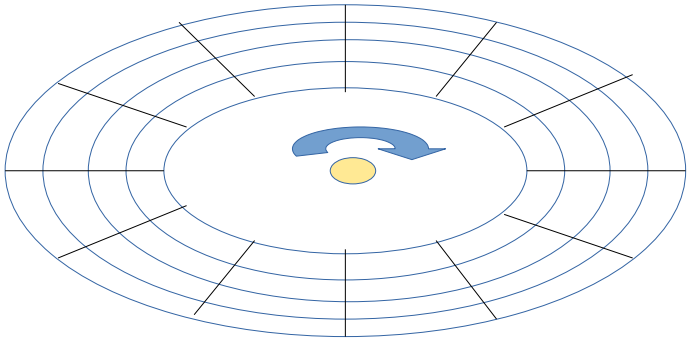
5- Les différents types de mémoires secondaires (MS)
principalement : disques magnétiques (ou disque durs HDD) et flash-disk (SSD...)
plus lents que **MC**, **non volatiles**, grande taille et moins coûteux que **MC**

6- Les mémoires d'archivage (tertiaire)
Généralement des bandes magnétiques (ou plus rarement des disques optiques)
Très lentes, **non volatile**, grande taille et généralement à très faible coût

Les Disques Durs (**HDD**)

C’est un dispositif de stockage externe (MS) relativement pas cher, robuste et offrant d’assez bonnes performances.





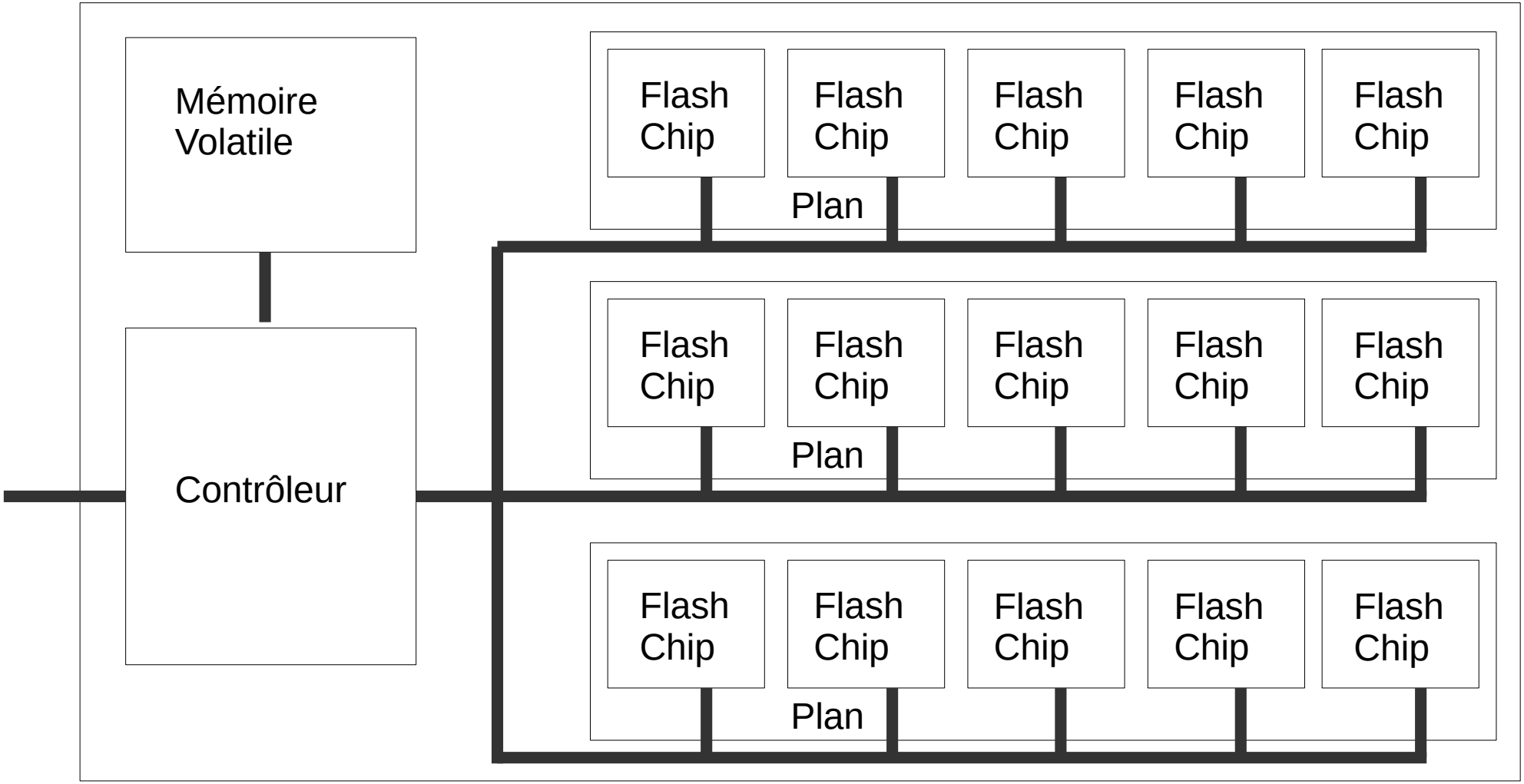
Exemple d’un disque de 1 TO

- 8 plateaux (donc 16 surfaces)
- 2^{16} , ou alors 65 536, pistes par surface
- (en moyenne) 2^{8} = 256 secteurs par piste.
- 2^{12} = 4096 octets par secteur.

La capacité du disque est le produit : 16 surfaces * 65,536 pistes * 256 secteurs * 4096 octets ⇒ 2^{40} octets (1 TO)

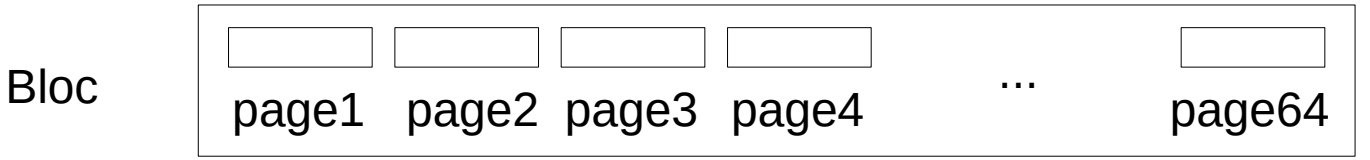
Les Disques à base de mémoires flash
Solid State Drive (**SSD**)

C’est un dispositif de stockage externe (MS) à base de mémoire flash NAND, robuste et offrant de meilleurs performances que le disque magnétique.



Les Disques à base de mémoires flash
Solid State Drive (**SSD**)

La mémoire est divisée en blocs (ex: 256KO) et chaque bloc est composé d'un certain nombre de pages (ex: 4KO)



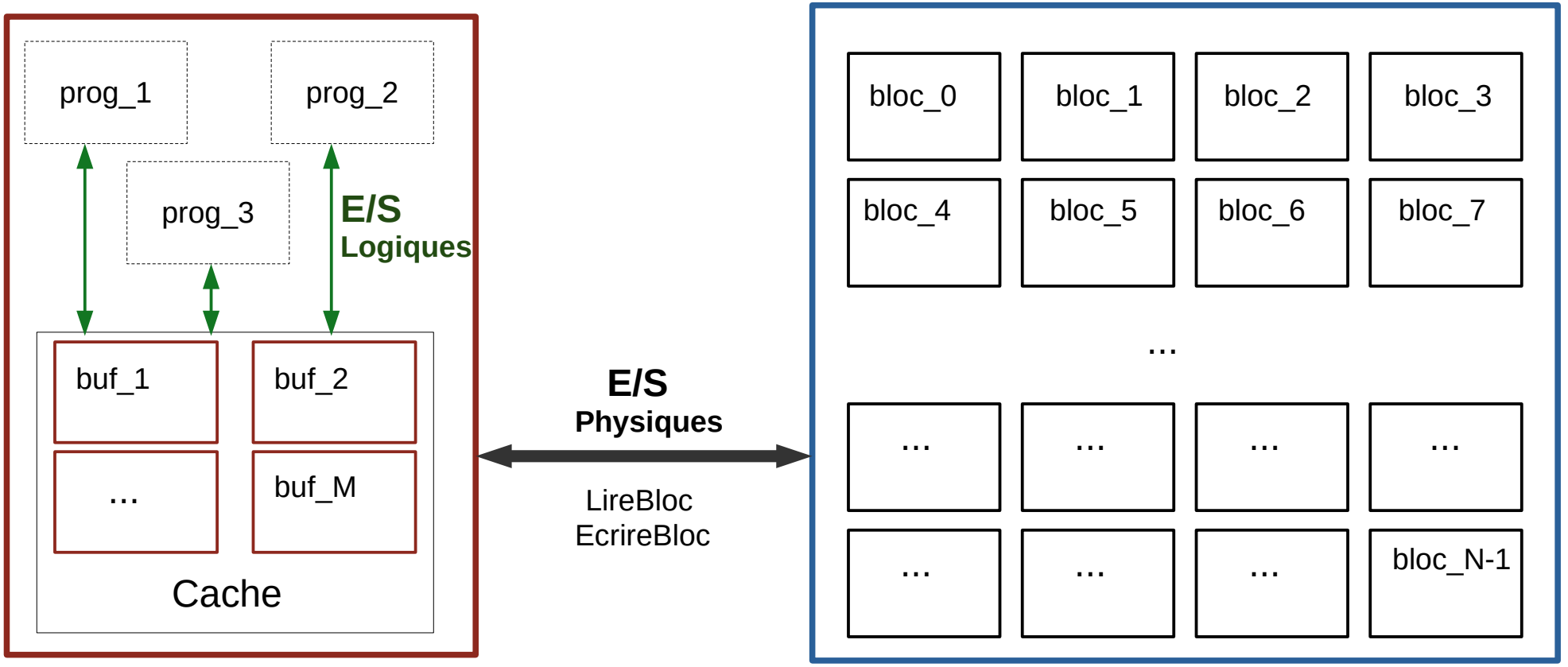
Il y a 3 opérations possibles :
- Lire une page ⇒ très rapide (20 microsecondes)
- Ecrire une page, à condition qu'elle soit dans un état effacé ⇒ rapide (100 à 200 microsecondes)
- Effacer toute les pages d'un bloc ⇒ lente (quelques millisecondes)

Les blocs supportent un nombre fixe d'effacements, au delà duquel ils deviennent inutilisables. Il faut donc répartir de manière uniforme les effacements sur l'ensemble des blocs du disque ⇒ C'est la technique du ***WearLeveling***

Lors d'une mise à jour du contenu d'une page, il est préférable d'écrire le nouveau contenu dans une nouvelle page (déjà effacée).
Les numéros de pages physiques doivent donc être cachés des utilisateurs
⇒ c'est le rôle de la ***FTL***

Interface d'accès pour la mémoire externe

Quelque soit le type du disque (HDD ou SSD) l'interface reste la même :
- L'unité de transfert = un bloc physique (1+ secteurs : HDD ou 1 page du SSD)



Mémoire Centrale (**MC**)

Mémoire Secondaire (**MS**)
HDD ou SSD

Notion de fichier

Niveau applicatif

Fichier typé :
Un fichier est une suite d'enregistrements (articles) en **MS**
Ex : F = < e1, e2, e3, … en >
Chaque enregistrement est formé d'un certain nombre de champs (attributs)
Ex : e = { nom :'aaa' , age : 19 , adr : 'cccccc' }

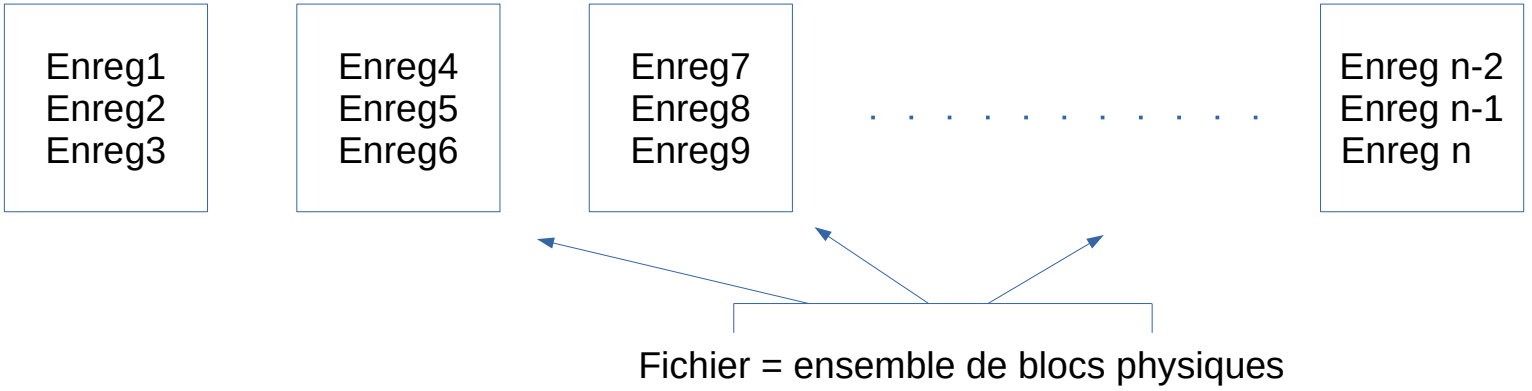
Fichier non typé (flux) :
Un fichier est une suite d'octets (caractères) en **MS**
Ex : F = < a,b,c,d,e,f,g,h,i,j,k,l,m, … >

Un fichier est une structure de données persistante hébergée dans un '*système de fichiers*' gérant la **MS**.
Les caractéristiques d'un fichier dépendent du système de fichier hôte, par exemple :
- nom externe
- taille (en octets)
- propriétaire
- droits d'accès
- dates de création, de dernière mise à jour, de dernière lecture ...

Notion de fichier

Niveau interne

Un fichier est un ensemble de blocs physiques en **MS**
Le contenu du fichier (les enregistrements ou le flux d'octets) est réparti sur les blocs alloués au fichier selon une certaine organisation.



Cette organisation définie la méthode d'accès utilisée pour lire et écrire le contenu du fichier.

Les caractéristiques du fichier sont sauvegardés soit dans des fichiers spéciaux (les répertoires) gérés par le systèmes d'exploitation soit au début de chaque fichier (bloc d'entête) des fois gérés par l'application elle-même.

Performances des structures de fichiers

Lors des opérations (traitement) sur les fichiers, le principal critère de performance se résume au nombre d'Entrée/Sortie physiques effectuées.

La complexité algorithmique est alors directement liée au nombres d'Entrée/Sortie.

Un autre critère de performance concerne l'occupation de l'espace mémoire

- L'encombrement mémoire :

⇒ Taille des structures de données utilisées / Taille des données stockées
≥ 100 % (plus le rapport est petit, plus c'est intéressant)

- Le facteur de chargement :

⇒ Nombre de données insérées / Nombre de place disponibles
Dans] 0 % , 100%] (plus le rapport est grand, plus c'est intéressant)

Les fichiers en langage C

Pour déclarer une variable fichier f (de type flux): ***FILE *f;***

Pour ouvrir un fichier: ***f = fopen(nomfichier, mode);***
Pour un fichier texte, le mode peut être :

mode	explications
"r"	ouverture en lecture
"w"	création d'un nouveau fichier (écrasement de l'ancien s'il existe déjà)
"a"	ouverture en mode « ajout »
"r+"	ouverture en lecture/écriture
"w+"	création d'un nouveau fichier en lecture/écriture
"a+"	ouverture en mode « ajout » en lecture/écriture

Si le mode contient le caractère '*b*' le fichier est ouvert en mode binaire
Ex : *"rb+" ou "r+b"* désigne une ouverture en lecture/écriture d'un fichier binaire

Pour fermer un fichier: ***fclose(f);***

Pour savoir si on a dépassé la fin de fichier (en mode lecture) : ***feof(f)***

Les fichiers en langage C

Le schéma général pour lire séquentiellement le contenu d'un fichier est le suivant :

```
FILE *f ;

f = fopen(...) ;

<< première_opération_de_lecture_de_données >>

while ( ! feof( f ) )  {

    // Traitement des données lus
    ...
    << prochaine_opération_de_lecture_de_données >>

}

fclose( f ) ;
```

Les fichiers en langage C

nbelt_lus = fread(buf, taille_elt, nb_elt, f);

lecture d'un nombre d'éléments consécutifs égal à nb_elt, chacun de taille taille_elt octets depuis la position courante du fichier f.
Le résultat de la lecture sera placé dans buf
La fonction retourne le nombre d'éléments effectivement lus

nbelt_ecr = fwrite(buf, taille_elt, nb_elt, f);

*Ecriture d'un nombre d'éléments égal à nb_elt, chacun de taille taille_elt octets dans le fichier f à partir de la position courante. Les octets à écrire (au nombre de taille_elt*nb_elt) sont récupérés depuis la zone pointée par buf.*
La fonction retourne le nombre d'éléments effectivement écrits

Après chaque opération de lecture ou d'écriture la position courante est implicitement modifiée du nombre d'octets effectivement transférés.

Pour modifier explicitement la position courante, on peut utiliser 'fseek' :
fseek(f, depl, origine) ;

Déplace la position courante d'un nombre d'octets égal à depl, relativement à :
- *début du fichier, si origine vaut SEEK_SET*
- *position courante, si origine vaut SEEK_CUR*
- *fin du fichier, si origine vaut SEEK_END*

Les fichiers en langage C

c = fgetc(f);

Lit et retourne (type int) le prochain caractère de f
Retourne EOF en cas de fin de fichier dépassée ou erreur.

fputc(c, f);

Ecrit le caractère c à la position courante du fichier f. Retourne EOF si erreur.

fgets(buf, n, f);

Lit dans buf, tous les caractères à partir de la position courante de f, jusqu'à trouver une fin de ligne '\n' (qui est aussi lue dans buf) ou alors jusqu'à ce que n-1 caractères soient lus. Un caractère de fin de chaîne '\0' est rajouté à la fin de buf.
En cas d'erreur ou fin de fichier, la fonction fgets retourne NULL.

fputs(buf, f);

Ecrit tous les caractères contenus de buf (sauf le '\0') dans le fichier f, à partir de la position courante. Pour écrire une ligne, il faut prévoir un caractère '\n' à la fin de buf (et avant le caractère de fin de chaîne '\0').
En cas d'erreur, la fonction retourne (type int) EOF.

Pour les E/S formatées :
fscanf(f, format, &var1, &var2, …) et ***fprintf(f, format, exp1, exp2, …)***

Machine abstraite pour les str. de fichiers

Lors de la déclaration d'un fichier, on définira le type des blocs, les Buffers, et les caractéristiques à utiliser (Entete)

Ex : TypeBloc = struct
tab : tableau[b] de TypeEnreg
NB : entier
Fin
F : FICHIER de TypeBloc BUFFER buf1, buf2 ENTETE (entier)

Dans cette déclaration on dit que :

- F est un fichier formé par des blocs ayant la structure TypeBloc
- buf1 et buf2 sont 2 variables (en MC) de type TypeBloc
  (on les utilisera comme buffers pour accéder aux blocs de F)
- Les caractéristiques de F se résume à un seul entier
  (on l'utilisera par exemple pour sauvegarder le nombre de blocs dans F)

On peut aussi déclarer les variables buffers indépendamment de F (dans des déclarations à part)

buf3 : TypeBloc ;

Les opérations du modèle sont :

OUVRIR(F , nomfichier , mode) Ouvre ou crée un fichier

mode = ‘A’ veut dire ouvrir en lecture/écriture un fichier qui existe déjà.
les caractéristiques seront lues en MC lors de l’ouverture.

mode = ‘N’ veut dire créer un nouveau fichier en lecture/écriture
les caractéristiques seront allouées en MC lors de le création

FERMER(F) Ferme le fichier. Les caractéristiques seront sauvegarder en MS

LireDir(F , i , buf) Lecture du bloc numéro i de F dans la variable buf

EcrireDir(F , i , buf) Ecriture de buf dans le bloc numéro i de F

ENTETE(F , i) Retourne la valeur de la caractéristique numéro i

AFF_ENTETE(F , i , v) Affecte v à la caractéristique numéro i

ALLOC_BLOC(F) Retourne le numéro d’un nouveau bloc alloué à F

Exemple

```
TypeBloc =   struct     tab : tableau[b] de TypeEnreg
                        NB : entier
             Fin
F : FICHIER de TypeBloc BUFFER buf ENTETE ( entier , entier )    /* nb_bloc , nb_enreg */
e : TypeEnreg ;
i, j, k, n : entier ;
Debut
    OUVRIR ( F , « fichier.dat », 'N' )  // un nouveau fichier
    Lire(n) ; i ← 1 ; j ← 1              // le remplir avec n enreg
    Pour k = 1,n
        Lire(e)                          // récupérer un enreg de l'entrée standard
        SI ( j ≤ b*0.7 )
                buf.tab[ j ] ← e ; j++   // et le mettre dans le buffer buf

        SINON                            // s'il n'y a pas de place dans buf,
                buf.NB ← j-1 ; ECRIREDIR( F , i , buf ) ;     // le sauvegarder en MS et
                buf.tab[1] ← e ; i++ ; j ← 2                  // remplir un nouveau
        FSI
    FP
    buf.NB ← j-1
    ECRIREDIR( F , i , buf)          // écriture du dernier bloc

    AFF_ENTETE( F , 1 , i ) ;  AFF_ENTETE( F , 2 , n ) ;     // m-a-j les caractéristiques
    FERMER( F )                                               // et fermer le fichier
Fin
```