

GESTION DES FICHIERS DE DONNEES AVEC LE LANGAGE C SOUS UNIX

Connaître et manipuler les différentes primitives de gestion de fichier de niveau haut sous UNIX.

Pré requis:

- ① Le langage C
- ② .Le système de gestion de fichiers d'UNIX (manipulation des fichiers avec le shell).

Plan

I/ Introduction

II/ Les catégories des fichiers en C

III/ Les primitives de manipulation des fichiers

1) Les primitives standards

2) Les primitives de manipulation : (jeu de procédures et de fonctions)

IV/ Conclusion

GESTION DES FICHIERS DE DONNEES AVEC LE LANGAGE C SOUS UNIX

I/ Introduction

Le langage C offre des primitives de gestion et de manipulation des fichiers de données.

Une différence de taille par rapport à certaines autres langages, en C, on ne différencie pas les fichiers séquentiels et les fichiers directs.

II/ Les catégories des fichiers en C

En langage C, on distingue deux catégories de fichier :

- * les fichiers standards (flux de données)
- * les fichiers systèmes (bas niveau)

Pour les fichiers standards et systèmes on distingue consécutivement :

- a) le flux texte : uniquement des caractères
- b) flux binaire : organisé en blocs d'octets contiguës. (On trouve soit des structures soit des tableaux).

Pour la manipulation de ces deux types de fichier, on dispose d'un jeu de procédures, de fonctions, et de bibliothèques spécifiques.

III/ Les primitives de manipulation des fichiers

1) Les primitives standards

Avant d'utiliser un fichier séquentiel, il faut :

- créer une mémoire tampon, dont laquelle seront enregistrées les informations avant leurs transferts de la mémoire centrale vers les fichiers de données.

La création se fait par :

```
FILE *Var_fpteur;
```

Structure pré
définie dans
stdio.h

Pointeur de flux qui
indique l'adresse de début
de la mémoire tampon

- Tout fichier doit être ouvert avant toute opération de création et d'utilisation (lecture / écriture)

Cette opération permet d'associer le nom du fichier à une variable qui est un pointeur (mémoire tampon) et de préciser le mode d'utilisation.

Pour l'ouverture :

```
Var_fpteur = fopen(nom_fichier, mode)
```

Mode	Signification
"r"	Ouverture d'un fichier existant en lecture seulement
"w"	Création et ouverture d'un fichier en écriture. Si le fichier existe, il est détruit et remplacé par le nouveau fichier créé (vide)
"a"	Ouverture d'un fichier existant pour lui ajouté en sortie des enregistrements. Si le fichier spécifier n'existe pas il est automatiquement créé.
"r+"	Ouverture en lecture / écriture d'un fichier existant.
"w+"	Création et ouverture d'un fichier. En cas d'existence, le fichier sera détruit.
"a+"	Ouverture en lecture / écriture d'un fichier, au quel on peut ajouter des enregistrements. En cas d'inexistence, le fichier sera créé.

- Tout fichier utilisé dans un programme, doit être fermé en fin de programme.

```
fclose(Var_fpteur);
```

2) Les primitives de manipulation : (jeu de procédures et de fonctions)

- forçage d'écriture de données dans un fichier

```
int fflush(FILE *ptr);
```

Cette fonction force écriture du buffert associer au fichier. On est certains que les information sont écrites sur le support.

- Détection de fin de fichier

```
int feof(FILE *ptr);
```

Cette fonction retourne une valeur non nulle après une lecture à la fin du fichier. Elle permet de distinguer après EOF entre l'erreur et le fin du fichier.

- Suppression d'un fichier

```
# include <io.h>
int unlink(char *path);
```

Cette fonction permet de détruire le fichier de nom path.

Exemple :

```
# include <io.h>
main()
{
    unlink("c:\travail\essai.txt");
}
```

- accès séquentiel :

```
int fgetc(FILE *fpt);
int getc(FILE *fpt);
```

Ces procédures permettent une lecture séquentielle d'un caractère sur le fichier fpt.

A l'issu de cette opération, la valeur retournée peut être un caractère du fichier ou EOF. S'il s'agit de EOF, alors c'est la fin de fichier ou erreur.

La fonction renvoie un entier et non un caractère. Cela permet de distinguer le EOF (-1) des autres caractères.

```
int fputc(char c,FILE *fpt);
int putc(char c,FILE *fpt);
```

Ecriture séquentielle du caractère c dans le fichier fpt. Ces fonctions retournent le code du caractère c dans le cas normal et EOF en cas d'erreur.

- Sorties formatées :

```
int fprintf(FILE *fpt, char *format, ...);
```

Cette fonction permet une écriture formaté dans le fichier fpt.

- Entrées formatées :

```
int fscanf(FILE *fpt, char *format, ...);
```

Cette fonction permet une lecture formaté dans le fichier fpt.

- Accès sélectif :

```
int fseek(FILE *fpt, long offset, int méthode);
```

Après l'ouverture du fichier, le pointeur est positionné sur le début du fichier ou sur la fin du fichier en mode append. L'entier long offset, positif ou négatif, spécifie le déplacement à effectuer à partir du point de départ indiqué par le paramètre méthode.

```
méthode  0 : à partir du début
          1 : à partir de la position courante
          2 : à partir de la fin.
```

Exemples:

1- Saisie d'une ligne de texte et sauvegarde dans un fichier:

```
#include<stdio.h>
main()
{
FILE  *fpt;
char  c;
fpt=fopen("fichier.dat","w");
do
    putc(c=getchar(),fpt);
while(c!='\n');
close(fpt);
}
```

2- Affichage d'une ligne de texte lue à partie d'un fichier.

```
#include<stdio.h>
main()
{
FILE  *fpt;
char  c;
if(fpt=fopen("fichier.dat","r")==NULL)
    printf(("Erreur d'ouverture du fichier"));
else  while((c=getc(fpt))!='\n')
        putchar(c);
}
```

- Lecture d'une chaîne de caractères:

```
char  *fgets(char  *s, int n, FILE  *fpt);
```

Cette fonction lit des caractères sur le fichier associé au "stream" fpt, les places dans la chaîne pointée par s. La lecture est stoppée lorsque n-1 caractères ont été lus.

- Lecture d'un entier à partir d'un fichier:

```
int getw(FILE *fpt);
```

Cette fonction permet de récupérer le prochain entier sur le fichier associé à fpt.

- Ecriture de chaîne de caractères:

```
int fputs(const char *s,FILE *fpt);
```

Cette fonction permet d'écrire la chaîne pointée par s dans le fichier associé au stream fpt. Le retour est constitué du dernier caractère transmis en cas de succès, si non EOF.

- Ecriture d'un entier dans un fichier:

```
int fputw(int w, FILE *fpt);
```

Cette fonction permet d'écrire l'entier w dans un fichier associé à fpt.

IV/ Conclusion