3ème ingénieur industriel électricité-électronique
option informatique
UE : Développement logiciel 1 –
AA : Développement en environnement Unix 1

(2022-2023)

T.P. n°1 : Les fichiers non bufferisés
Objectifs : • Maîtriser la notion de fichiers non bufferisés • Manipulation des fonctions classiques d'accès : open, close, read, write, lseek
A présenter le 27/10 pendant la séance de labo (pondération EC : 70% (donc projet 35%))

Le but de l'exercice est de créer une petite application qui va gérer un fichier d'enregistrements de taille fixe. Il s'agit d'un fichier du type « dictionnaire » dont chaque enregistrement correspond

- à une **clé** (le « mot ») → chaine de caractères
- à une **valeur** (la « définition » associée au mot) → chaine de caractères

Dans le fichier, un enregistrement sera représenté par la structure

```
typedef struct enreg
{
  int  valide ;
  char cle[20];
  char valeur[40];
} S_ENREG ;
```

comprenant la clé (dont la taille ne devra pas passer dépasser 19 caractères), la valeur (dont la taille ne devra pas dépasser 39 caractères), et un flag « valide » (valant 1 ou 0) précisant si l'enregistrement est valide ou pas (s'il a été supprimé).

1) Création de la librairie d'accès au fichier

Dans un premier temps, on vous demande de créer la librairie « Hash » et de la mettre sous la forme de deux fichiers : **Hash.cpp** (définitions des fonctions) et **Hash.h** (déclaration de la structure + prototypes des fonctions).

La librairie « Hash » devra contenir les fonctions suivantes :

int ajout(const char* nomFichier, const char* cle, const char* valeur)

Cette fonction devra

1. ouvrir le fichier en écriture (si celui-ci n'existe pas, il sera créé),
2. se positionner en fin de fichier,
3. remplir correctement une structure S_ENREG en fonction des paramètres reçus (valide = 1),
4. écrire la structure remplie sur disque,
5. fermer le fichier,
6. retourner 0 si tout s'est bien passé et -1 si une erreur s'est produite.

int suppression(const char* nomFichier, const char* cle)

Cette fonction devra

1. ouvrir le fichier en lecture/écriture (s'il n'existe pas, retour -1),
2. se déplacer séquentiellement dans le fichier jusqu'à trouver le bon enregistrement. S'il est trouvé, le flag « valide » devra être mis à zéro dans le fichier (il s'agit d'une suppression logique, et non physique, de l'enregistrement),
3. fermer le fichier,
4. retourner 1 si l'enregistrement a été trouvé (et donc supprimé), sinon retour 0.

int recherche(const char* nomFichier, const char* cle, char* valeur)

Cette fonction devra

1. ouvrir le fichier en lecture seule (s'il n'existe pas, retour -1),
2. se déplacer séquentiellement dans le fichier jusqu'à trouver le bon enregistrement recherche sur la clé). S'il est trouvé (avec un flag « valide » = 1), la chaine de caractères « valeur » lue dans le fichier sera mise à l'adresse valeur reçue en paramètre dans la fonction,
3. fermer le fichier,
4. retourner 1 si l'enregistrement (valide) a été trouvé, sinon retour 0.

int modification(const char* nomFichier, const char* cle, const char* valeur)

La modification ne concerne que la valeur et non la clé qui ne change pas.

Cette fonction devra

1. ouvrir le fichier en lecture/écriture (s'il n'existe pas, retour -1),
2. se déplacer séquentiellement dans le fichier jusqu'à trouver le bon enregistrement. S'il est trouvé, la nouvelle valeur sera mise à jour dans le fichier,
3. fermer le fichier,
4. retourner 1 si l'enregistrement a été trouvé (et donc modifié), sinon retour 0.

int liste(const char* nomFichier)

Cette fonction devra

1. ouvrir le fichier en lecture seule (s'il n'existe pas, retour -1),
2. se déplacer séquentiellement dans le fichier et afficher tous les couples clé/valeur valides trouvés dans le fichier.
3. fermer le fichier,
4. retourner 0.

2) Création de l'application utilisant la librairie

On vous demande de créer un programme qui

- reçoit en paramètre en ligne de commande le nom du fichier à utiliser,
- propose un menu dans lequel il sera possible à l'utilisateur d'ajouter/supprimer/rechercher/modifier/lister des enregistrements dans le fichier.

Ce menu tournera en boucle jusqu'à ce que l'utilisateur choisisse l'option « Quitter ».

3) Création du makefile

Après avoir rangé vos fichiers sources dans des sous-répertoires, par exemple :

./TP1/Hash/Hash.cpp
./TP1/Hash/Hash.h
./TP1/Applic.cpp
./TP1/makefile

On vous demande de créer le makefile permettant de tout compiler automatiquement.

4) BONUS (non obligatoire donc) : Nettoyage du fichier

On se rend bien compte qu'au fur et à mesure que l'on ajoute/supprime des enregistrements, il va y avoir de plus en plus d'enregistrements non valides qui prennent inutilement de la place sur disque et ralentissent l'exécution du programme. On vous demande donc

1. d'ajouter à la librairie une fonction « nettoyage » qui reçoit en paramètre le nom du fichier source à « nettoyer » et le nom d'un nouveau fichier qui sera créé en supprimant tous les enregistrements non valides.
2. de créer un second programme recevant, en paramètres en ligne de commande, le nom du fichier à nettoyer et le nom du fichier destination. Ce programme appellera la fonction « nettoyage » de la librairie.
3. de modifier le makefile afin que la librairie et les deux exécutables soient compilés automatiquement.