

API DIET file management system

22 février 2010

1 Présentation

La pile logicielle *DIET file management system* est fournie avec une bibliothèque de développement permettant à des programmes en C ou C++ de tirer partie de l'ensemble de ses possibilités. Cette bibliothèque est compilée de manière optionnelle en même temps que les programmes composant l'ensemble logiciel. Pour activer la compilation de la bibliothèque, l'utilisateur passera à ON l'option de compilation `DEV_LIB` définie dans la configuration *cmake* du projet. Une option de compilation supplémentaire est fournie pour faciliter le débogage des applications utilisant la bibliothèque : il s'agit de l'option `DEBUG_LIB` qui fournit des indications sur les erreurs rencontrées aux développeurs.

L'activation de la compilation de la bibliothèque ajoutera la librairie *diet-fms-lib* au répertoire *lib* situé à la racine du répertoire d'installation du projet ainsi qu'un fichier *diet-fms-lib.h* dans le sous répertoire *include* de ce dernier.

2 Types et structures de données

L'API définit un type énuméré `diet_file_type_t` désignant le type d'un fichier parmi les suivants :

- `DFMS_BLOCK` : fichier spécial de type bloc.
- `DFMS_CHARACTER` : fichier de type caractère.
- `DFMS_DIRECTORY` : répertoire.
- `DFMS_SYMLINK` : lien symbolique.
- `DFMS_SOCKET` : socket.
- `DFMS_FIFO` : fichier FIFO.
- `DFMS_REGULAR` : fichier régulier.

Une structure de données regroupant différentes informations à propos d'un fichier est également définie. Il s'agit de la structure `diet_stat_t` définie comme suit :

```
struct diet_stat_t {
    char path[PATH_MAX];           /*! The file path. */
    char owner[_POSIX_LOGIN_NAME_MAX]; /*! The owner of the file. */
    char group[_POSIX_LOGIN_NAME_MAX]; /*! The group of the file. */
    mode_t perms;                  /*! The access permissions of the file. */
    uid_t uid;                     /*! The owner numeric UID. */
    gid_t gid;                     /*! The group numeric GID. */
    size_t size;                   /*! The file size (in bytes). */
    time_t atime;                  /*! The last acces time. */
    time_t mtime;                  /*! The last modification time. */
    time_t ctime;                  /*! The last inode change time. */
}
```

```

diet_file_type_t type;                                /*! The file type. */
};

```

Le tableau de caractères **path** contient le nom du fichier. Le tableau **owner** contient le login de l'utilisateur à qui le fichier appartient. Le tableau de caractères **group** contient le groupe du fichier. Le champs **perms** de type **mode_t** est le masque de bits représentant les droits du propriétaire du fichier, des membres de son groupe et des autres utilisateurs sous le format classique POSIX. Le champ **uid** de type **uid_t** contient l'identifiant numérique du propriétaire du fichier et le champ **gid** de type **gid_t**, l'identifiant numérique de son groupe. Le champs **size** contient la taille du fichier en octets. Les champs **atime**, **mtime** et **ctime** de type **time_t** contiennent respectivement la date du dernier accès au fichier, la date de sa dernière modification et la date de la dernière modification de son inode. Enfin, le champ **type** de type **diet_file_type_t** définit plus haut, indique le type du fichier.

3 Fonctions de modification des attributs d'un fichier

Deux fonctions de l'API permettent de changer les attributs d'un fichier local ou distant :

- **int dietchgrp(const char* path, const char* group)**
Fonction permettant de changer le groupe d'appartenance du fichier. Son chemin d'accès est passé en premier paramètre sous la forme **[host:]path**. Le nom du nouveau groupe d'appartenance est passé en deuxième paramètre. Il sera éventuellement transcrit en un groupe différent sur la machine distante si celle-ci définit une table de correspondance des identifiants de groupes globaux et locaux.
- **int dietchmod(const char* path, const mode_t mode)**
Fonction permettant de changer les droits d'accès à un fichier. Son chemin d'accès est passé en premier paramètre sous la forme **[host:]path**. Les nouveaux droits d'accès sont passés en second paramètre utilisant une valeur de type POSIX **mode_t**.

Les deux fonctions renvoient une valeur différente de zéro si l'opération a échoué. Si l'option **DEBUG_LIB** a été activée lors de la compilation de l'API, elles affichent également un message décrivant l'erreur rencontrée sur la sortie d'erreur standard.

4 Fonctions d'affichage du fichier

L'API propose deux fonctions permettant d'afficher respectivement les *n* premières et *n* dernières lignes d'un fichier. Il s'agit des fonctions :

```

int diethead(const char* path, const unsigned int nline,
             char* buffer, const size_t maxsize)

int diettail(const char* path, const unsigned int nline, char* buffer,
             const size_t maxsize);

```

Le premier paramètre désigne le chemin d'accès au fichier sous la forme **[host:]path**. Le paramètre **nline** indique le nombre de lignes à afficher. L'utilisateur passe un tampon dont la mémoire est allouée et une taille maximale utilisable pour stocker le résultat. Attention, le passage d'une valeur plus grande que la taille du tampon pour ce dernier paramètre peut provoquer des erreurs imprévisibles.

Les deux fonctions renvoient une valeur différente de zéro si l'opération a échoué. Si l'option **DEBUG_LIB** a été activée lors de la compilation de l'API, elles affichent également un message décrivant l'erreur rencontrée sur la sortie d'erreur standard.

5 Informations sur les fichiers

Deux fonctions de l'API permettent d'obtenir des informations sur les fichiers distants ou locaux :

```
int dietls(const char* path, char* buffer, const size_t maxsize);
```

```
int dietstat(const char* path, struct diet_stat_t* buf);
```

La fonction `dietls` récupère sous forme “brute” la liste des noms de fichiers et répertoires du répertoire dont le chemin est passé en paramètre sous la forme `[host:]path`. Les noms de fichiers et sous-répertoires sont stockés sous forme de chaînes de caractères séparées par des espaces dans le tampon fourni par l'utilisateur. Le dernier paramètre permet de préciser une taille maximale pour la chaîne résultat copiée dans le tampon.

La fonction `dietstat` stocke dans la structure `buf` de type `struct diet_stat_t*` passée par l'utilisateur, l'ensemble des informations recueillies par le système à propos du fichier considéré.

Les deux fonctions renvoient une valeur différente de zéro si l'opération a échoué. Si l'option `DEBUG_LIB` a été activée lors de la compilation de l'API, elles affichent également un message décrivant l'erreur rencontrée sur la sortie d'erreur standard.

6 Fonctions de création et suppression de répertoires

L'API dispose de deux fonctions permettant respectivement la création et la suppression de répertoires :

```
int dietmkdir(const char* path, const mode_t mode);
```

```
int dietrmdir(const char* path);
```

La fonction de création de répertoire `dietmkdir` prend en paramètre le chemin du répertoire à créer ainsi que ses permissions d'accès d'origine.

La fonction de suppression de répertoire prends en paramètre le seul chemin d'accès au fichier.

Les deux fonctions renvoient une valeur différente de zéro si l'opération a échoué. Si l'option `DEBUG_LIB` a été activée lors de la compilation de l'API, elles affichent également un message décrivant l'erreur rencontrée sur la sortie d'erreur standard.

7 Suppression d'un fichier

L'API permet également la suppression d'un fichier par l'intermédiaire de la fonction `dietrm` dont le prototype est le suivant :

```
int dietrm(const char* path);
```

Elle prend en paramètre le chemin d'accès au fichier sous la forme `[host:]path`.

La fonction renvoie une valeur différente de zéro si l'opération a échoué. Si l'option `DEBUG_LIB` a été activée lors de la compilation de l'API, elle affiche également un message décrivant l'erreur rencontrée sur la sortie d'erreur standard.

8 Copie et déplacement de fichiers

Quatre fonctions permettent la copie ou le déplacement d'un fichier par l'intermédiaire de l'API : deux fonctions de copie dont l'une est en mode synchrone et l'autre asynchrone et deux fonctions de déplacement une pour chaque mode également.

8.1 Copie de fichier

Les deux fonctions permettant la copie d'un fichier sont les suivantes :

```
int dietcp(const char* src, const char* dest);

int dietcp_async(const char* src, const char* dest,
                 unsigned long long* thrRef);
```

La fonction `dietcp` prends simplement les chemins d'accès au fichier et à la destination de la copie désirée sous la forme `[host:]path`. Un troisième paramètre permettant d'obtenir un identifiant du transfert est nécessaire pour la version asynchrone de la copie. Il sera utilisé avec la fonction `dietwait` permettant d'attendre la fin d'un transfert asynchrone.

8.2 Déplacement de fichier

Les deux fonctions permettant le déplacement d'un fichier sont les suivantes :

```
int dietmv(const char* src, const char* dest);

int dietmv_async(const char* src, const char* dest,
                 unsigned long long* thrRef);
```

La fonction `dietmv` prends simplement les chemins d'accès au fichier et à sa destination désirée sous la forme `[host:]path`. Un troisième paramètre permettant d'obtenir un identifiant du transfert est nécessaire pour la version asynchrone du déplacement. Il sera utilisé avec la fonction `dietwait` permettant d'attendre la fin d'un transfert asynchrone.

8.3 Attente de la fin d'un transfert

L'API fournit des fonctions de transferts asynchrones. Afin d'attendre l'achèvement de ces transferts, on utilisera la fonction `dietwait` dont le prototype est le suivant :

```
int dietwait(const unsigned long long thrID);
```

Cette fonction prend en paramètre un identifiant de transfert obtenu des fonctions de transferts asynchrones `dietcp` ou `dietmv`.

9 Exemples d'utilisation de l'API

Dans cette section nous présentons un programme d'exemple d'utilisation de l'API. Ce programme prend en paramètre un fichier de configuration DIET, un chemin d'accès à un fichier et le nom d'hôte d'une machine distante. Afin de simplifier sa lecture, il est conçu de manière très simple et n'effectue notamment pas tous les contrôles qu'il serait nécessaire d'effectuer. Le fichier de configuration DIET est passé en premier paramètre, le chemin d'accès au fichier est passé en deuxième paramètre sous la forme `[host:]path` et le nom d'hôte distant est passé en troisième paramètre.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <DIET_client.h>
#include <diet-fms-lib.h>

int main(int argc, char* argv[]) {
    char* tmpstr;
    char* lsresult;
    diet_initialize(argv[1], argc, argv);

    /* Construct the name "<host>:test_dir" */
    tmpstr = calloc(strlen(argv[3])+strlen(":test_dir")+1, 1);
    strcpy(tmpstr, argv[3]);
    strcat(tmpstr, ":test_dir");

    /* Creates a directory on remote host. */
    /* 511 in decimal = 777 in octal : */
    /* Owner, group and others can access */
    /* the directory for reading, writing */
    /* and executing. */
    dietmkdir(tmpstr, 511);

    /* Copies the file into this remote */
    /* directory. */
    dietcp(argv[2], tmpstr);

    /* Allocates a buffer for dietls */
    lsresult = malloc(1024);

    /* Lists the remote directory. */
    dietls(tmpstr, lsresult, 1024);
    printf("The remote dir %s contains the files : %s\n",
        tmpstr, lsresult);

    return EXIT_SUCCESS;
}

```