Légende : Zohra Rabab

Opendir

La fonction opendir() ouvre un flux répertoire correspondant au répertoire *nom*, et renvoi**e un pointeur** sur ce flux. Le flux est positionné sur la première entrée du répertoire.

Exemple d'utilisation :

```c
#include <stdio.h>
#include <stdlib.h>
#include <dirent.h>  // Pour opendir, readdir, closedir
#include <errno.h>   // Pour gérer les erreurs

void open_directory(const char *path) {
    DIR *dir = opendir(path);  // Ouvre le répertoire
    if (dir == NULL) {
        // Si l'ouverture échoue, on affiche une erreur
        perror("Erreur lors de l'ouverture du répertoire");
        return;
    }

    struct dirent *entry;
    // Parcourt les fichiers dans le répertoire
    while ((entry = readdir(dir)) != NULL) {
        printf("%s\n", entry->d_name);  // Affiche le nom du fichier ou dossier
    }

    closedir(dir);  // Ferme le répertoire
}

int main() {
    const char *path = "./";  // Exemple de répertoire courant
    open_directory(path);     // Appel de la fonction pour afficher les fichiers
    return 0;
}
```

Readdir:

readdir est une fonction qui permet de lire le contenu d'un dossier (répertoire). Imagine un dossier sur ton ordinateur : il contient des fichiers et peut-être d'autres dossiers. readdir sert à lister tout ce qui est dedans

Exemple d’utilisation :

```c
#include <stdio.h>
#include <dirent.h>

int main() {
    DIR *d = opendir("."); // Ouvre le dossier courant
    struct dirent *file;

    if (d) {
        while ((file = readdir(d)) != NULL) {
            printf("%s\n", file->d_name); // Affiche chaque fichier/dossier
        }
        closedir(d); // Ferme le dossier
    }
    return 0;
}
```

Closedir

ferme le flux de répertoire associé à *dir*. Après cette invocation, le descripteur *dir* du flux de répertoire n'est plus disponible. renvoie 0 si elle réussit, ou -1 si elle échoue, auquel cas *errno* contient le code d'erreur.

Exemple d'utilisation : Combinaison Opendir et Closedir

```c
#include <stdio.h>
#include <dirent.h>  // Pour opendir, closedir

int main() {
    // Ouvre le répertoire courant
    DIR *dir = opendir(".");
    if (dir == NULL) {
        printf("Erreur d'ouverture du répertoire.\n");
        return 1;
    }

    // Code qui ferait quelque chose avec le répertoire (par exemple, lire
    // Mais ici on se concentre juste sur l'utilisation de closedir()

    // Ferme le répertoire avec closedir()
    closedir(dir);
    printf("Répertoire fermé avec succès.\n");

    return 0;
}
```

Stat :

`stat` est une fonction en C qui permet d'obtenir des **informations sur un fichier ou un dossier** :

La taille du fichier

La date de dernière modification

Le type (fichier, dossier, lien...)

Les permissions d'accès

Exemple d’utilisation :

```c
#include <stdio.h>
#include <sys/stat.h>

int main() {
    struct stat fileStat;  // Structure pour stocker les infos
    if (stat("monfichier.txt", &fileStat) == 0) {  // Récupère les infos
        printf("Taille du fichier: %ld octets\n", fileStat.st_size);
        printf("Dernière modification: %ld\n", fileStat.st_mtime);
    } else {
        perror("Erreur avec stat");  // Affiche une erreur si le fichier n'ex
    }
    return 0;
}
```

lstat :

Est utilisée pour obtenir des informations sur un fichier ou un répertoire, en particulier des informations liées aux attributs du fichier, comme son type, ses permissions, sa taille, etc. Elle est similaire à la fonction stat(), mais avec une différence importante : lstat() ne suit pas les liens symboliques, tandis que stat() suit les liens symboliques pour obtenir les informations du fichier cible.

Exemple d'utilisation :

```c
#include <stdio.h>
#include <sys/stat.h>
#include <unistd.h>

int main() {
    const char *path = "./exemple";  // Le fichier ou répertoire à tester

    struct stat statbuf;

    // Utiliser lstat pour obtenir les informations du fichier
    if (lstat(path, &statbuf) == -1) {
        perror("Erreur lors de l'appel à lstat");
        return 1;
    }

    // Vérifier si c'est un répertoire, un fichier ordinaire ou un lien symbolique
    if (S_ISDIR(statbuf.st_mode)) {
        printf("%s est un répertoire.\n", path);
    } else if (S_ISREG(statbuf.st_mode)) {
        printf("%s est un fichier ordinaire.\n", path);
    } else if (S_ISLNK(statbuf.st_mode)) {
        printf("%s est un lien symbolique.\n", path);
    } else {
        printf("%s est un autre type de fichier.\n", path);
    }

    return 0;
}
```

Getpwuid

getpwuid est une fonction en C qui permet d'obtenir des informations sur un utilisateur du système à partir de son UID (User ID).

Elle est utile pour :
Trouver le nom d'utilisateur associé à un UID
Vérifier le chemin du répertoire personnel (/home/user)
Obtenir des infos sur l'utilisateur comme son shell par défaut (/bin/bash)

Exemple d’utilisation :

```c
#include <stdio.h>
#include <pwd.h>
#include <unistd.h>

int main() {
    uid_t uid = getuid(); // Récupère l'UID de l'utilisateur actuel
    struct passwd *pw = getpwuid(uid); // Obtient les infos de l'utilisateur

    if (pw) {
        printf("Nom d'utilisateur : %s\n", pw->pw_name);
        printf("Répertoire personnel : %s\n", pw->pw_dir);
        printf("Shell par défaut : %s\n", pw->pw_shell);
    } else {
        perror("Erreur avec getpwuid");
    }

    return 0;
}
```

Getgrgid:

La fonction renvoie un pointeur vers une structure contenant les champs de l'enregistrement de la base de données de groupe correspondant au GID *gid*.

Exemple d’utilisation :

```c
#include <stdio.h>
#include <grp.h>      // Pour getgrgid()
#include <stdlib.h>   // Pour exit()

int main() {
    gid_t gid = 1000;  // Exemple de GID (change-le avec un autre GID si nécessaire)

    // Utiliser getgrgid() pour récupérer les informations du groupe
    struct group *grp = getgrgid(gid);
    if (grp == NULL) {
        perror("Erreur lors de l'appel à getgrgid");
        exit(1);
    }

    // Afficher le nom du groupe
    printf("Le groupe avec GID %d s'appelle : %s\n", gid, grp->gr_name);

    return 0;
}
```

Getcwd

getcwd est une fonction en C qui permet de récupérer le **répertoire courant** (aussi appelé **dossier de travail actuel**) dans lequel le programme s'exécute.

Exemple d’utilisation :

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    char cwd[1024];  // Tableau pour stocker le chemin du répertoire courant
    if (getcwd(cwd, sizeof(cwd)) != NULL) {  // Récupère le répertoire courant
        printf("Répertoire courant : %s\n", cwd);  // Affiche le chemin
    } else {
        perror("Erreur avec getcwd");  // Affiche une erreur en cas de problèm
    }
    return 0;
}

```

Ctime()

La fonction convertit une valeur de type time_t qui représente le temps écoulé en secondes en une chaîne de caractères lisible par l'homme, représentant cette date et heure.

```c
#include <stdio.h>
#include <time.h>

int main() {
    time_t current_time;
    current_time = time(NULL);  // Obtient l'heure actuelle

    // Convertir l'heure en une chaîne lisible
    char *time_str = ctime(&current_time);
    printf("L'heure actuelle : %s", time_str);

    return 0;
}
```

Free() :

La fonction est utilisée pour libérer un bloc de mémoire précédemment alloué dynamiquement à l'aide des fonctions comme `malloc()`, `calloc()`, ou `realloc()`. Cette fonction permet de retourner la mémoire allouée au système, afin qu'elle puisse être réutilisée, évitant ainsi des fuites de mémoire.

Exemple d’utilisation :

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    // Allouer de la mémoire pour un entier
    int *ptr = (int *)malloc(sizeof(int));  // Allocation dynamique

    if (ptr == NULL) {
        printf("Erreur d'allocation mémoire !\n");
        return 1;
    }

    // Utiliser la mémoire allouée
    *ptr = 42;  // Stocker une valeur à l'adresse pointée par ptr
    printf("La valeur de ptr est : %d\n", *ptr);

    // Libérer la mémoire allouée
    free(ptr);  // Libération de la mémoire

    // Après la libération, il est prudent de définir ptr à NULL
    ptr = NULL;

    // Assurez-vous de ne pas utiliser le pointeur après qu'il ait été libéré
    // printf("La valeur de ptr après free est : %d\n", *ptr);  // Ceci est dangereux !

    return 0;
}
```

Malloc :

malloc (pour memory allocation) est une fonction en C qui permet d'allouer de la mémoire dynamique pendant l'exécution du programme. Cela signifie qu'on peut réserver de l'espace mémoire pour des données à un moment donné, et non pas à la compilation.

Exemple d’utilisation :

```c
#include <stdio.h>
#include <stdlib.h>  // Nécessaire pour malloc

int main() {
    int *ptr = malloc(5 * sizeof(int));  // Alloue de la mémoire pour un table

    if (ptr == NULL) {  // Vérifie si l'allocation a échoué
        perror("Erreur d'allocation mémoire");
        return 1;
    }

    // Remplir et afficher les valeurs du tableau
    for (int i = 0; i < 5; i++) {
        ptr[i] = i * 10;  // Stocke des valeurs dans le tableau
        printf("ptr[%d] = %d\n", i, ptr[i]);
    }

    free(ptr);  // Libère la mémoire allouée
    return 0;
}
```