Les fichiers :

un processus peut être confronté à différents problèmes :

- taille de ce que manipule ce processus
- persistance des informations
- communication ("basique") entre différents processus

Types des fichiers :

- fichiers ASCII : lignes de textes eventuellement modifiable par un éditeur
- fichiers binaires : (ex : pdf, jpeg, .o ...) a besoin d'éditeur spécifique pour l'ouvrire

Nom du fichier :

abstraction des données techniques qui permettent de référencer les données sur le disque

Unix :

l'exclusion est indicative mais pas imposée

Dos\Windows :

l'exclusion est définie par le système :

.c → code c

.jpg → image

...

Accès aux fichiers :

Accès séquentiel
- lecture depuis le début vers la fin

Accès aléatoire
- accès directe sur le fichier

Dans tous les accès aujourd'hui se fait de manière aléatoire

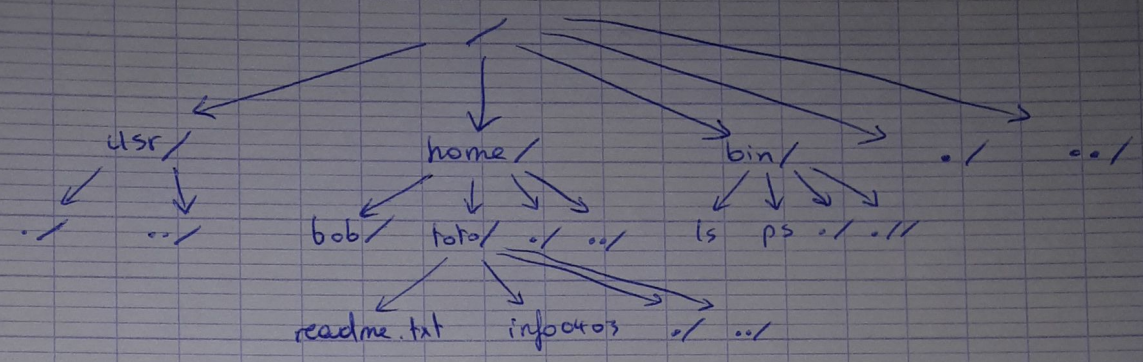
Attributs des fichiers :

information complémentaires d'un fichier (dépendant du SE)
- protection sur le fichier
- créateur du fichier
- propriétaire du fichier
- date de création
- taille
- date de modification
- ...

Repertoires :

intérêt : - hirerarchiser
- organiser

On a une arborescence de fichiers :



Partitionnement et partition :

Le disque s'organise comme suit :

Master boot record	table de partition	Partition 1	...	Partition n

bloc du boot	super bloc	gestion de l'espace libre	i node	rep. racine	reps. et fichiers

Implentation des fichiers :

- contigüe : → perte de temps à l'accès
  → perte d'espace lors de la suppression

- aléatoire : → solution utilisée

- blocs de taille fixée

mémoirésation des blocs libres → tableau de bits / liste chainée

Ouverture d'un fichier :

int open (const char * path, int flag, mode_t mode)

ouvre le fichier et retourne le descripteur de fichier

le fichier ouvert a le nom path,
flag peut prendre les valeurs :

- O_RDONLY → lecture seule
- O_WRONLY → écriture seule
- O_RDWR → lecture écriture
- O_APPEND → concaténation
- O_CREATE → créer le fichier s'il existe pas

retourne valeur négatif si le fichier ne s'ouvre pas

mode : droits sur le fichier 0644

Fermeture d'un fichier :

int close (int fd)

ferme le fichier dont le descripteur de fichier est fd

un fichier ouvert a une tête de lecture associée pour une ouverture ou lecture/écriture, à l'ouverture, cette tête est positionné en début de fichier

Lecture sur un fichier :

size_t read (int fd, void * buf, size_t nboctets)

tent de lire le fichier dont le descripteur est fd, nb octets et place le contenu des octets effectivement lus dans le buffer buf. read retourne le nb d'octets effectivement lus.

Ecriture sur un fichier :

size_t write (int fd, const void * buf, size_t nboctets)

tente d'écrire nboctets octets de l'espace mémoire buff sur le fichier référencé par fd. write retourne le nb d'octets effectivement écrits.

Exercice :

écrire un programme qui copie un fichier source dans un fichier destinataire

```
int main (int argc, char * argv[]) {
    int src, dest;
    int nblu;
    char c;    // char c[255];

    if (argc != 3) {
        printf ("Arguments pas corrects.");
        exit (-1);
    }
```

```
if((src = open(argv[1], O_RDONLY, 0) <= 0) {
        printf("argument pas valable.");
        exit(-1);
}
if((dest=open(argv[2], O_WRONLY|O_Create, 744))<=0){
        printf("argument pas valable.");
        exit(-1);
}
                                        x     255     255
while ((nblu = read(src, (void *) &c, 1)) == -1)
        write(dest, (void *)&c, 1);
} write(dest, (void *)c,  x   nblu
close(src);      nblu);
close(dest);
exit(0);
}
```

tête de lecture :

positionner la tête de lecture à une position dans un fichier

⇒ commande lseek

off_t lseek (int d, off_t offset, int whence)

→ retourne le nombre d'octets dont la tête de lecture sera deplacée sur le fichier référencé par fd

on indique l'origine du déplacement (where) et l'ampleur du déplacement (avec l'offset en nombre d'octets)

call func

fwhence :

SEEK_SET : début du fichier
SEEK_CUR : position courante
SEEK_END : fin de fichier

Suppressions d'un fichier :

int unlink (const char * path)
détruit le lien entre le nom de fichier path et les données dans le disque

Information sur un fichier :

→ fonction stat
int stat (const char * path, struct stat * buf)
- recupère une structure strat
- associé au fichier path.

```
struct stat {
    dev_t     st_dev;     // périf sur le que et stocké le fichier
    ino_t     st_ino;     // n° inode
    mode_t    st_mode;    // protection
    nlink_t   st_nlink;   // nb de liens matériels
    uid_t     st_uid;     // uid propriétaire
    gid_t     st_gid;     // gid ———————
    dev_t     st_rdev;    // type du périphérique
    off_t     st_size;    // taille en octets
    blksize_t st_blksize; // taille du bloc E/S
    blkcnt_t  st_blks;    // nb de blocs
    time_t    st_atime;   // heure dernier accès
```

```
        time_t  st_mtime ;  // heure dernier modif
        time_t st_ctime ;  //heure dernier changement
            /* modif = modifier le contenu
               changement = changement de attributs */
}
```

Ecrire un programme qui affiche le uid du prop., la taille, et la date du dernier changement d'un fichier passé en paramètre :

```
int main (int argc, char* argv[]) {
int stat;>
struct ot stat
        if (argc != 2) {
            printf ("Pb avec les param");
            exit(-1);
        }

        if (stat = stat(argv[1], &a) >= 0) {
            printf ("%s" a.st_uid, a.st_size, a. ...)
        }
}

int main (int argc, char * argv[]) {
    struct stat * buff = (struct stat)* malloc (sizof
    // test passage de param                (struct stat));
    stat (argv[1], buff);
    printf ("fichier %s : \n
            \t Prop. : %d \n
            \t taille : %d \n
            \t date : %d \n", buff->st_uid,
            buff->st_size, buff->st_ctime);
    exit(0);
}
```

Manipulation des répertoires :

```
DIR * opendir (const char * path)
int closedir ( DIR * dirp )
struct dirent * readdir (DIR * dirp)
void rewinddir ( DIR * dirp )

struct dirent {
    ...
    char d_name [256];
    ...
}
```

Ecrire un programme qui affiche le nom des fichiers dans le repertoires courant.

```
int main (int argc, char *argv [] ){
    DIR * p;
    struct dirent * de;
    p = opendir ("./");
    while ((de = readdir (p)) != NULL ) {
        printf ("%s\n", de -> d_name);
    }
    closedir (p);
    exit (0);
}
```

Pareil mais afficher la taille des fichiers :

```
int main (int argc, char * argv [] ) {
    DIR * p; struct stat buf;
    struct dirent * de;
    p = opendir (".");
    while ((de = readdir (p)) != NULL) {
        printf ("%s\n", de-> de_name);
        stat ( de -> d_name, &buf);
        printf ("\t %s\n", buff.st_size);
    }
    closedir (p);
    exit(0);
}
```

<u>Rmq</u> pour savoir si une entrée (dirent) est un repertoire :

```
S_ISDIR (mode_t m)   // rep ?
S_ISREG (mode_t m)   // fichier regulier ?
S_ISLINK (_______)   // lien ?
S_ISFIFO (_______)   // tube fifo ?
S_ISSOCK (_______)   // socket ?
```

Exo :

Ecrire un programme qui affiche le nom des repertoires dans le rep courant :

```
int main (int argc, char * argv[]) {
    DIR * p;
    struct dirent * de;
    struct stat buf;
    p = opendir (".");
    while ((de = readdir(p)) != NULL) {
        stat ( de->d_name, &buf);
        if (!_ISDIR ( buf.st_mode))
            printf ("%s\n", de->d_name);
        }
    } closedir (p);
    exit(0);
}
```

```
char * getcwd (char * buf, size_t size)
```
→ copie le chemin absolu du repertoire courant de travail dans buf de taille size

```
int chdir (cast char * path)
```
→ change le repertoire courant par path

```
int rmdir (cast char * path)
```
→ supprime le repertoire path (vide)

Ecrire un programme qui affiche le nom des fichiers et repertoires contenus dans le rep courant ou dans un de ses sous repertoires

⚠ gérer le cas de . et ..