Cours 4 : Entrées / Sorties

- **Primitives d'entrées-sorties POSIX**
  `unistd.h, sys/stat.h, sys/types.h, fcntl.h`
  - Constituent l'interface avec le noyau Unix (**appels systèmes**)
    permettent l'utilisation des services offerts par le noyau.
  - Portabilité des programmes sur Unix.

- **Bibliothèque d'entrées-sorties standard C**
  `stdio.h`
  - + grand niveau de portabilité : indépendance du système.
  - Surcouche d'optimisation (eg. suite d'appels à *write*)
    accès asynchrones, bufferisés et formatés (type).

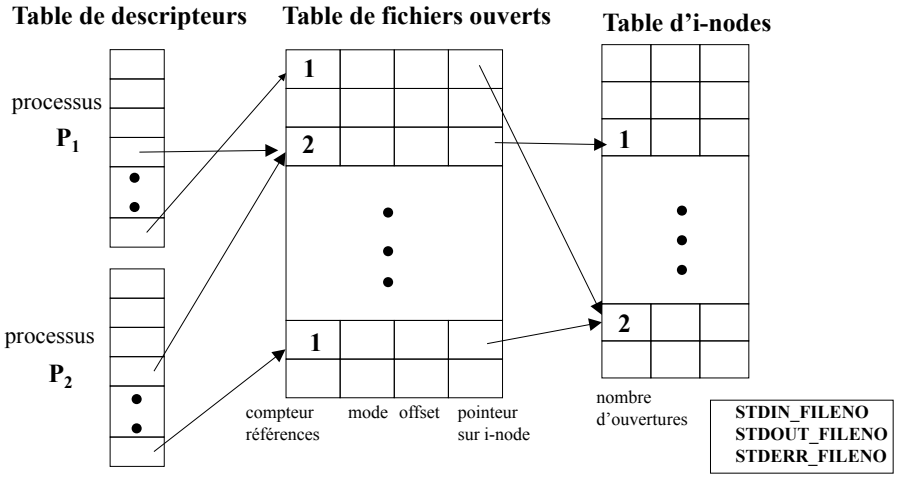
Fichiers d'en-tête

- **Types de base universels (= portables).**
  *eg. FILE**
- **Constantes symboliques.**
  eg. NBBY (8)
- **Structures et types utilisés dans le noyau.**
  eg. struct stat
- **Prototypes des fonctions.**
  *eg. FILE *fopen(const char *, const char *);*

Quelques constantes de configuration (POSIX)

- **LINK_MAX**
  nb max de liens physiques par i-node (8).
- **PATH_MAX**
  longueur max pour le chemin (nom) d'un fichier (255).
- **NAME_MAX**
  longueur max des noms de liens (14).
- **OPEN_MAX**
  nb max d'ouvertures de fichiers simultanées par processus (16).

Organisation des Tables



Quelques erreurs associées aux E/S

#include <errno.h>

extern int errno;

- **EACCESS** : accès interdit.
- **EBADF**: descripteur de fichier non valide.
- **EEXIST** : fichier déjà existant.
- **EIO**: erreur E/S.
- **EISDIR**: opération impossible sur un répertoire.
- **EMFILE**: trop de fichiers ouverts pour le processus (> OPEN_MAX).
- **EMLINK** : trop de liens physiques sur un fichier (> LINK_MAX).
- **ENAMETOOLONG** : nom fichier trop long (>PATH_MAX)
- **ENOENT** : fichier ou répertoire inexistant.
- **EPERM** : droits d'accès incompatible avec l'opération.

Consultation de l'i-node (stat)

- **Structure stat** <sys/stat.h>

```
struct stat {
        dev_t       st_dev;     /* device file resides on */
        ino_t       st_ino;     /* the file serial number */
        mode_t      st_mode;    /* file mode */
        nlink_t     st_nlink;    /* number of hard links to the file*/
        uid_t       st_uid;     /* user ID of owner */
        gid_t       st_gid;     /* group ID of owner */
        dev_t       st_rdev;    /* the device identifier*/
        off_t       st_size;     /* total size of file, in bytes */
        unsigned long  st_blksize; /* blocksize - file system I/O*/
        unsigned long  st_blocks;  /* number of blocks allocated */
        time_t      st_atime;   /* file last access time */
        time_t      st_mtime;   /* file last modify time */
        time_t      st_ctime;   /* file last status change time */
}
```


Type de fichier

Champ *st_mode* de *struct stat*
Type : masque S_IFMT (POSIX : macros)

- **Fichiers réguliers : données (S_IFREG)**
  - macro: S_ISREG (t)
- **Répertoires (S_IFDIR)**
  - macro: S_ISDIR (t)
- **Tubes FIFO (S_FIFO)**
  - macro: S_ISFIFO (t)
- **Fichiers spéciaux : périphs bloc (S_IFBLK) ou caractère (S_IFCHR)**
  - macro: S_ ISBLK (t) et S_ ISCHR (t)
- **Liens symboliques (S_IFLNK)**
  - macro: S_ISLNK (t)
- **Sockets (S_IFDOOR)**
  - macro: S_ISSOCK (t)

Droits d'accès

- **Propriétaire, groupe et autres** **(Champ *st_mode* de *struct stat*)**
  - lecture, écriture et exécution

	Propriétaire	**Groupe**	**Autres**
Lecture	S_IRUSR	S_IRGRP	S_IROTH
Ecriture	S_IWUSR	S_IWGRP	S_IWOTH
Exécution	S_IXUSR	S_IXGRP	S_IXOTH
Les trois	S_IRWXU	S_IRWXG	S_IRWXO

> ls - l **rwxr-xr--**

S_IRWXU| S_IRGRP | S_IXGRP| S_IROTH

Fonctions de consultation de l'i-node

- **Obtention des caractéristiques d'un fichier**
  - **int stat(const char *file_name, struct stat *buf);**
  - **int fstat(int fdes, struct stat *buf);**
    - Résultats récupérés dans une *struct stat*
- **Test des droits d'accès d'un processus sur un fichier**
  - **int access (const char* pathname, int mode);**
    - mode : **R_OK**, **W_OK**, **X_OK**, **F_OK**
      (droit de lecture, écriture, exécution, existence).

Exemple - stat

```
#define _POSIX_SOURCE 1
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>

int main (int argc, char* argv []) {
 struct stat stat_info;

 if ( stat (argv[1], &stat_info) == -1)
   { perror ("erreur stat");
    return EXIT_FAILURE;
   }

 if (S_ISDIR (stat_info.st_mode) )
    printf ("fichier répertoire\n");

 printf ("Taille fichier : %ld\n", (long)stat_info.st_size);

 if (stat_info.st_mode & S_IRGRP)
   printf ("les usagers du même goupe peuvent lire le fichier\n");

 return EXIT_SUCCESS;
}
```


Manipulation de liens physiques

- **Création d'un lien physique sur un répertoire**
  - **int link (const char *origine, const char *cible)**
    - permet de créer un nouveau lien physique
    - contraintes
      - *origine* ne peut pas être un répertoire
      - *cible* ne doit pas exister

```
> ln Fic1 Fic2
> ls -ia
```

24	.
43	..
78	Fic1
78	Fic2

- **Suppression d'un lien physique**
  - **int unlink (const char *ref)**
    - supprime le lien associé à *ref*
    - fichier supprimé si:
      - nombre de liens physiques sur le fichier est nul
      - nombre d'ouvertures du fichier est nul
- **Changement de nom de lien physique**
  - **int rename (const char *ancien, const char *nouveau)**
    - *nouveau* ne doit pas exister
    - impossible de renommer . et ..

code renvoi : 0 (succès) ; -1 (erreur)

Changement d'attributs d'un i-node

- **Droits d'accès**
  - **int chmod (const char* reference, mode_t mode);**
  - **int fchmod (int descripteur, mode_t mode);**
    attribution des droits d'accès *mode* au fichier :
    - de nom *reference*
    - associé à *descripteur*
- **Propriétaire**
  - **int chown (const char* reference, uid_t uid, gid_t gid);**
  - **int fchown (int descripteur, uid_t uid, gid_t gid);**
    modification du propriétaire *uid* et du groupe *gid* d'un fichier

code renvoi : 0 (succès) ; -1 (erreur)

Exemple - chmod

test-chmod.c

```
#define _POSIX_SOURCE 1
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdlib.h>

int main (int argc, char* argv []) {
   if (chmod (argv[1], S_IRUSR | S_IWUSR| S_IRGRP | S_IWGRP | S_IROTH |
     S_IWOTH) == 0)
        printf ("fichier %s en lecture-ecriture pour tous les usagers \n ", argv[1]);
  else { perror ("chmod");
         return EXIT_FAILURE;
       }
   return EXIT_SUCCESS;
}
```

```
>ls –l fich1
  -rw------- ….
>test-chmod fich1
  -rw-rw-rw- …..
```


Primitives de base (1)

- **Ouverture d'un fichier : open**
  - **int open (const char* reference, int flags);**
  - **int open (const char* reference, int flags, mode_t droits);**
    - renvoie un numéro de descripteur
    - **flags**: – **O_RDONLY** : ouverture en lecture
      – **O_WRONLY** : ouverture en écriture
      – **O_RDWR** : ouverture en lecture-écriture
      – **O_CREAT**: création d'un fichier s'il n'existe pas
      – **O_TRUNC**: vider le fichier s'il existe
      – **O_APPEND** : écriture en fin de fichier
      – **O_SYNC** : écriture immédiate sur disque
      – **O_NONBLOCK** : ouverture non bloquante
    - **droits**: lecture, écriture, exécution

code renvoi :
descripteur (succès)
-1 (erreur)

Primitives de base (2)

- **Fermeture de fichier : close**
  - **int close (int descripteur);**
    - Ferme le descripteur correspondant à un fichier en désallouant son entrée de la table des descripteurs du processus.
    - Si nécessaire, mise à jour table des fichiers et table des i-nodes.
- **Création d'un fichier**
  - **int creat (const char* reference, mode_t droits);**
    correspond à l'appel suivant:
    open (reference, int flags, O_WRONLY | O_CREAT | O_TRUNC, droits);

Primitives de base (3)

- **Lecture dans un fichier : `read`, `readv`, `pread`**
  - **ssize_t read (int desc, void* tampon, size_t nbr);**
    - Demande de lecture d'au + *nbr* caractères du fichier correspondant à *desc.*
    - Les caractères lus sont écrits dans *tampon.*
    - Renvoie le nombre de caractères lus ou -1 en cas d'erreur.
    - La lecture se fait à partir de la **position courante**
      offset de la *Table des Fichiers Ouverts ;* mise à jour après la lecture.
  - **ssize_t readv (int desc, const struct iovec* vet, int n);**
    - Données récupérées dans une *struct iovec* de taille *n.*

```
struct iovec {
   void *iov_base;
   size_t iov_len; }
```

  - **ssize_t pread (int desc, void* tampon, size_t nbr, off_t pos);**
    - Lecture à partir de la position *pos* ; *offset* n'est pas modifié.

Primitives de base (4)

- **Ecriture dans un fichier : `write`, `writev`, `pwrite`**
  - **ssize_t write (int desc, void* tampon, size_t nbr);**
    - Demande d'écriture de *nbr* caractères contenus à partir de l'adresse *tampon* dans le fichier correspondant à *desc*.
    - Renvoie le nombre de caractères écrits ou -1 en cas d'erreur.
    - L'écriture se fait à partir de la fin du fichier (O_APPEND) ou de la position courante.
    - Modifie le champ *offset* de la *Table des Fichiers Ouverts.*
  - **ssize_t writev (int desc, const struct iovec* vet, int n);**
  - **ssize_t pwrite (int desc, void* tampon, size_t nbr, off_t pos);**

Exemple – open, read et write

test-rw.c

```
#define _POSIX_SOURCE 1
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

#define SIZE_TAMPON 100
char tampon [SIZE_TAMPON];
int main (int argc, char* argv []) {
  int fd1, fd2;    int n,i;

  fd1 = open (argv[1], O_WRONLY|O_CREAT|
              O_SYNC,0600);
  fd2 = open (argv[1], O_RDWR);

  if ( (fd1== -1) || (fd2 == -1)) {
     printf ("open %s" ,argv[1]);
    return EXIT_FAILURE;
}
```

```
  if (write (fd1,"abcdef", strlen ("abcdef")) == -1) {
     perror ("write");
     return EXIT_FAILURE;
  }
  if (write (fd2,"123", strlen ("123") ) == -1) {
     perror ("write");
     return EXIT_FAILURE;
  }
  if ((n= read (fd2,tampon, SIZE_TAMPON)) <=0) {
      perror ("fin fichier\n");
      return EXIT_FAILURE;
  }
  for (i=0 ; i<n; i++)
    printf ("%c",tampon [i]);

  return EXIT_SUCCESS;
}
```


Primitives de base (5)

- **Manipulation de l'offset: `lseek`**
  - **off_t lseek (int desc, off_t position, int origine);**
    - Permet de modifier la position courante *(offset*) de l'entrée de la *Table de Fichiers Ouverts* associée à *desc*.
    - La position courante prend comme nouvelle valeur : *position* + *origine.*
    - **origine**:
      - **SEEK_SET**: 0 (début du fichier)
      - **SEEK_CUR** : Position courante
      - **SEEK_END** : Taille du fichier
    - Renvoie la nouvelle position courante ou -1 en cas d'erreur.

Exemple – lseek

test-lseek.c

```
#define _POSIX_SOURCE 1
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>

#define SIZE_TAMPON 100
  char tampon [SIZE_TAMPON];
int main (int argc, char* argv []) {
 int fd1, fd2;    int n,i;

 fd1 = open (argv[1], O_WRONLY|O_CREAT|
             O_SYNC,0600);
 fd2 = open (argv[1], O_RDWR);

 if ( (fd1== -1) || (fd2 == -1)) {
   printf ("open %s" ,argv[1]);
   return EXIT_FAILURE;
}
```

```
  if (write (fd1,"abcdef", strlen ("abcdef")) == -1) {
     perror ("write");
     return EXIT_FAILURE;
  }
 if (write (fd2,"123", strlen ("123") ) == -1) {
     perror ("write");
     return EXIT_FAILURE;
 }
/* déplacement au début du fichier */
 if (lseek(fd2,0,SEEK_SET) == -1) {
   perror ("seek");
   return EXIT_FAILURE;
}
 if ((n= read (fd2,tampon, SIZE_TAMPON)) <=0) {
    perror ("fin fichier\n");
    return EXIT_FAILURE; }
 for (i=0 ; i<n; i++)
    printf ("%c",tampon [i]);
 return EXIT_SUCCESS;
}
```

Exemple – fork

test-fork.c

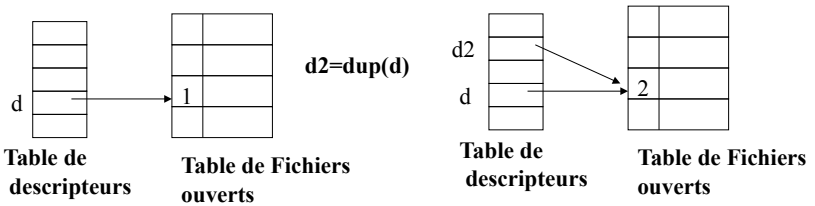
```
#define _POSIX_SOURCE 1
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <sys/wait.h>

#define SIZE_TAMPON 100
 char tampon [SIZE_TAMPON];

 int main (int argc, char* argv []) {
   int fd1, fd2;   int n,i;
  if ((fd1 = open (argv[1], O_RDWR| O_CREAT |
              O_SYNC,0600)) == -1) {
       perror ("open \n");
      return EXIT_FAILURE;
}
if (write (fd1,"abcdef", strlen ("abcdef")) == -1) {
   perror ("write");
   return EXIT_FAILURE; }
if (fork () == 0) {
   /* fils */
     if ((fd2 = open (argv[1], O_RDWR)) == -1) {
        perror ("open \n");
        return EXIT_FAILURE;
     }
     if (write (fd1,"123", strlen ("123")) == -1) {
        perror ("write");
        return EXIT_FAILURE;
     }
     if ((n= read (fd2,tampon, SIZE_TAMPON)) <=0) {
        perror ("fin fichier\n");
        return EXIT_FAILURE;
     }
     for (i=0 ; i<n; i++)
        printf ("%c",tampon [i]);
     exit (0);
 }
 else /* père */
   wait (NULL);
 return EXIT_SUCCESS;
}
```


Duplication de descripteur

- **La primitive dup**
  - **int dup (int desc);**
    - Recherche le + petit descripteur disponible dans la table des descripteurs du processus et en fait un synonyme de *desc*.
  - **int dup2 (int desc, int desc2);**
    - Force le descripteur *desc2* à devenir synonyme de *desc*.



Exemple – dup2

```
#define _POSIX_SOURCE 1

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>
int fd1;
int main (int argc, char* argv []) {

 if ((fd1 = open (argv[1], O_WRONLY| O_CREAT,0600)) == -1) {
     perror ("open \n");
     return EXIT_FAILURE;
  }

 printf ("avant le dup2: descripteur %d \n", fd1);
 dup2 (fd1, STDOUT_FILENO);
printf ("après le dup2 \n");

 return EXIT_SUCCESS;
}
```

Redirection de stdout

> test-dup2 fich5

avant le dup2 : descripteur 3

> cat fich5

après le dup2

Liens symboliques

- **int symlink (const char* reference, const char* lien);**
  - créer un lien symbolique sur le fichier *reference*
- **int lstat (const char* reference, struct stat* pStat);**
- **ssize_t readlink (const char* ref, char* tampon, size_t taille);**
  - récupère à l'adresse *tampon* la valeur du lien symbolique (son contenu)
- **lchmod (const char* reference, mode_t mode);**
- **lchown (const char* reference, uid_t uid, gid_t gid);**

Les entrées-sorties sur répertoires (1)

- **Ouverture d'un répertoire**
  - **DIR * opendir (const char* reference);**
    - ouvre en lecture le répertoire de référence *reference*
    - lui alloue un objet du *type DIR*, dont l'adresse est renvoyée au retour
- **Lecture d'une entrée**
  - **struct dirent* readdir (DIR *pDir);**
    - lit l'entrée courante du répertoire associé à *pDir*
    - place le pointeur sur l'entrée suivante
    - renvoie un pointeur de type *struct dirent*

```
struct dirent {
   …
   char d_name [];
}
```

    - renvoie NULL en cas d'erreur ou lorsque la fin de fichier est atteinte

Les entrées-sorties sur répertoires (2)

- **Rembobinage**
  - **void rewinddir (DIR *pDir);**

    Repositionne le pointeur des entrées associé à *pDir* sur la 1re entrée dans le répertoire.
- **Fermeture**
  - **int closedir (DIR *pDir);**

    Ferme le répertoire associé à *pDir*.

    Les ressources allouées au cours de l'appel à *opendir* sont libérées.

Exemple – Lister le contenu d'un répertoire

test-listdir.c

```
#define _POSIX_SOURCE 1
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <dirent.h>

DIR *pt_Dir;
struct dirent* dirEnt;
```

```
int main (int argc, char* argv []) {
 if ( ( pt_Dir = opendir (argv[1]) ) == NULL) {
    perror ("opendir");
     return EXIT_FAILURE;
 }
  while ((dirEnt= readdir (pt_Dir)) !=NULL)
    printf ("%s\n", dirEnt->d_name);
 closedir (pt_Dir);
 return EXIT_SUCCESS;
}
```

```
>test-listdir rep1
   .
   ..
   fich1
   fich2
```


Ecritures dans les répertoires

- **Création d'un répertoire**
  - **int mkdir (const char* ref, mode_t mode);**
    - Création d'un répertoire vide en spécifiant les droits.
- **Supprimer un répertoire**
  - **int rmdir (const char* ref);**
- **Renommer un répertoire**
  - **int rename (const char* ancien, const char* nouveau);**

code renvoi : 0 (succès) ; -1 (erreur)

Obtention/modification des attributs d'un fichier

- **Fonction fcntl**
  - Permet de modifier des attributs associés à un descripteur
    - mode d'ouverture
    - duplication du descripteur
    - verrouillage des zones du fichier
  - Signature dépend de la valeur de `cmd`
    - `int fcntl(int fd, int cmd);`
    - `int fcntl(int fd, int cmd, long arg);`
    - `int fcntl(int fd, int cmd, struct flock *lock);`

Obtention/modification des attributs d'un fichier

- **Fonction fcntl**
  - **Argument *`cmd`* :**
    - F_GETFD : obtenir la valeur des attributs du descripteur
    - F_SETFD : modifier les attributs du descripteur
    - F_GETFL : obtenir la valeur des attributs du mode d'ouverture
    - F_SETFL : modifier le mode d'ouverture
      `O_APPEND, O_NONBLOCK, O_NDELAY, O_SYNC`
    - F_DUPFD : duplication de descripteur dans le + petit descripteur disponible
      `fcntl (fd, F_DUPFD, 0);` est équivalent à *`dup (fd);`*
    - F_SETLK, F_SETLKW, F_GETLK : verrouillage
  - **Code de retour :**
    - F_DUPFD : le nouveau descripteur, sinon -1 en cas d'erreur (voir errno)
    - F_GETFD, F_GETFL: valeur des attributs, sinon -1 en cas d'erreur (voir errno)
    - F_SETFD, F_SETFL : 0 en cas de succès, -1 en cas d'erreur (voir errno)

Verrou sur fichier

- **Fonction fcntl**
  Le verrouillage est consultatif !
  Verrouillage effectif ⇒ sémaphores
  - **Type de verrou**
    - Exclusif (write) : aucun autre processus ne peut verrouiller une zone avec verrou exclusif
    - Partagé (read) : tout autre processus peut verrouiller une zone avec verrou partagé
  - **`struct flock`**

```
off_t l_start;    /* starting offset */
off_t l_len;      /* len = 0 means until end of file */
pid_t l_pid;      /* lock owner */
short l_type;     /* lock type: F_RDLCK, F_WRLCK, F_UNLCK */
short l_whence;   /* type of l_start */
```

  - **Argument *`cmd`***
    F_GETLK : récupérer le verrou courant sur la zone définie par `lock`
    F_SETLK : (dé)verrouiller la zone définie par `lock`
    F_SETLKW : idem, mais attente bloquante jusqu'à ce que la zone soit libérée d'autres verrous

Exemple – Fonction fcntl

fcntl-test.c

```
#define _POSIX_SOURCE 1
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <fcntl.h>

int main (int argc, char* argv []) {
 int fd1, mode_ouv;

 if ((fd1 = open (argv[1], O_RDWR)) == -1){
  perror ("open"); return EXIT_FAILURE;
 }

 if (write (fd1, "abc", 3) == -1) {
  perror ("write"); return EXIT_FAILURE;
 }

mode_ouv = fcntl (fd1, F_GETFL);
 mode_ouv |= O_APPEND;
 fcntl (fd1,F_SETFL, mode_ouv);

  if (write (fd1, "xyz", 3) == -1) {
   perror ("write"); return EXIT_FAILURE;
  }

  close (fd1);
  return EXIT_SUCCESS;
}
```

La bibliothèque E/S standard C

- **Constitue une couche au-dessus des appels système correspondant aux primitives de base d'E/S POSIX.**
- **But : travailler dans l'espace d'adressage du processus**
  - E/S dans des tampons appartenant à cet espace d'adressage
  - Objet de type FILE, obtenu lors de l'appel à la fonction *fopen* :
    - permet de gérer le tampon associé au fichier
    - possède le numéro du descripteur du fichier
      - **STDIN_FILENO = stdin**
      - **STDOUT_FILENO = stdout**
      - **STDERR_FILENO = stderr**
  - `fflush` force l'écriture du contenu du tampon dans les caches système

La bibliothèque E/S standard C

- **Fichier <stdio.h>**

  Constantes:
  - **NULL** : adresse invalide
  - **EOF** : reconnaissance de fin de fichier
  - **FOPEN_MAX**: nb max de fichiers manipulables simultanément
  - **BUFSIZ**: taille par défaut des tampons

La bibliothèque E/S standard C

- **Fichier <stdio.h>**
  - **Types:**
    - **FILE**: type dédié à la manipulation d'un fichier
      Gère le tampon d'un fichier ouvert.
    - **fpos_t**: position dans un fichier
    - **size_t**: longueur du fichier
  - **Objets prédéfinis de type FILE*:**
    - **stdin:** objet d'entrée standard
    - **stdout** : objet de sortie standard
    - **stderr :** objet de sortie-erreur standard

Fonctions de base (1)

- **Ouverture d 'un fichier**
  - **FILE* fopen (const char*reference, const char *mode);**
    - Arguments
      - *reference* chemin d'accès au fichier
      - *mode* mode d'ouverture
    - Renvoie un pointeur vers un objet *FILE* associé au fichier, NULL si échec.
    - Association d'un *tampon* pour les lectures/écritures, et d'une *position courante*.
    - *mode*:

`r`	**lecture seulement.**
`r+`	**lecture et écriture sans création ou troncature du fichier.**
`w`	**écriture avec création ou troncature du fichier.**
`w+`	**lecture et écriture avec création ou troncature du fichier.**
`a`	**écriture en fin de fichier ; création si nécessaire.**
`a+`	**lecture et écriture en fin de fichier ; création si nécessaire.**

Fonctions de base (2)

- **Nouvelle ouverture d'un fichier**
  - **FILE* freopen (const char* reference, const char *mode, FILE* pFile);**
    - Associe à un objet déjà alloué une nouvelle ouverture.
    - Redirection d'E/S.
    - **Exemple: Redirection sortie standard**
      - freopen ("fichier1","w", stdout);
- **Obtention du descripteur associé à l'objet FILE**
  - **int fileno (FILE* pFile);**
- **Obtention d'un objet du type FILE à partir d'un descripteur.**
  - **FILE *fdopen (const int desc, const char *mode);**
    - Le *mode* d'ouverture doit être compatible avec celui du descripteur.

Exemple – fdopen

fdopen-test.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>

int main (int argc, char ** argv) {

int fd;
FILE *ptFile;
 if ( (fd = open (argv[1], O_RDWR |
  O_CREAT)) == -1) {
  perror ("open"); exit (1);
}

 if ((ptFile = fdopen (fd,"w+")) == NULL) {
  perror ("fdopen"); exit (1);
}

 if (write (fd,"ab",2)==-1) {
    perror ("write");exit (1);
 }

  if (fputs ("cd",ptFile) == -1) {
     perror ("fputs");exit (1);
  }
  return (EXIT_SUCCESS);
  }
```


Fonctions de base (3)

- **Test de fin de fichier**
  - **int feof (FILE *pFile);**
    - associé aux opérations de lecture
    - renvoie une valeur ≠ 0 si la fin de fichier associée à *pFile* a été détectée
- **Test d'erreur**
  - **int ferror (FILE *pFile);**
    - renvoie une valeur ≠ 0 si une erreur associée à *pFile* a été détectée
- **Fermeture d'un fichier**
  - **int fclose(FILE *pFile);**
    - ferme le fichier associé à *pFile*.
    - Transfert de données du tampon associé.
    - Libération de l'objet *pFile*.
    - Renvoie 0 en cas de succès et EOF en cas d'erreur.

Gestion du tampon

- **A chaque ouverture de fichier**
  tampon de taille BUFSIZ est automatiquement alloué
- **Association d'un nouveau tampon:**
  - **int setvbuf(FILE *pFile, char* tampon, int mode, size_t taille);**
    - Permet d'associer un nouveau tampon de taille *taille* à *pFile*.
    - Critère de vidage (*mode*)
      - _IOFBF: lorsque le tampon est plein
      - _IOLBF: lorsque le tampon contient une ligne ou est plein
      - _IONBF: systématiquement
- **Vidage du tampon**
  - **int fflush (FILE *pFile);**
    Si *pFile* vaut NULL, tous les fichiers ouverts en écriture sont vidés

Fonctions de base (4)

- **Lecture**
  - **Un caractère**
    - **int fgetc (FILE* pFile);**
      - retourne le caractère suivant du fichier sous forme entière
        EOF en cas d'erreur ou fin de fichier
      - **int getchar (void)** équivalent à **fgetc(stdin)**;
  - **Une chaîne de caractères**
    - **char *fgets (char *pChaine, int taille, FILE* pFile);**
      - lit au + *taille-1* éléments de type *char* à partir de la *position courante* dans *pFile*
      - arrête la lecture si *fin de ligne (\n*, incluse dans la chaîne)
        ou *fin de fichier* est détectée
      - renvoie NULL en cas d'erreur ou fin de fichier
        - Test avec *feof* ou *ferror.*

Exemple fgetc et fgets

fgetc-s-test.c

```
#define POSIX_SOURCE 1
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#define TAILLE_BUFF 100

int main (int argc, char ** argv) {
 char c;
 char buff[TAILLE_BUFF];
 FILE *ptLire;

 if ( (ptLire = fopen (argv[1], "r")) == NULL) {
  perror ("fopen"); exit (1);
 }
```

```
/* lecture d'un caractère */
   if ((c=fgetc(ptLire))!= EOF)
      printf ("%c",c);

 /* lecture d'une chaîne */
     if (fgets (buff,TAILLE_BUFF, ptLire)
       !=NULL)
            printf("%s\n",buff);

fclose (ptLire);
  return (EXIT_SUCCESS);
}
```


Fonctions de base (5)

- **Lecture (cont)**
  - **lecture d'un tableau d'objets**

    size_t fread (void *p, size_t taille, size_t nElem, FILE* pFile);
    - Lit au + *nElem* objets à partir de la position courante dans *pFile*.
    - Tableau des objets lus sauvegardé à l'adresse *p*.
    - Chaque objet est de taille *taille.*
    - Retourne le nombre d'objets lus
      0 en cas d'erreur ou fin fichier (test *feof* ou *ferror).*

Fonctions de base (6)

- **Lecture (cont)**
  - **lecture formatée**

    int fscanf (FILE* pFile, const char *format, …);
    - Lit à partir de la *position courante* dans le fichier pointé par *pFile.*
    - *format* : procédures de conversion à appliquer aux suites d'éléments de type *char* lues.
    - **scanf** équivaut à **fscanf** sur **stdin**.
    - Retourne le nombre de conversions réalisées ou EOF en cas d'erreur.

Exemple – scanf

```
#define _POSIX_SOURCE 1
#include <stdio.h>
#include <stdlib.h>

 int ret, var_int;
char var_char, var_string[10];

int main (int argc, char* argv []) {

 ret = scanf ("%c %s %d", &var_char, var_string, &var_int);
 if (ret == 3)
  printf ("var_char=%c, var_string = %s, var_int = %d\n", var_char,
           var_string,var_int);
 else{
    fprintf (stderr, "erreur: ret = %d\n", ret);  return EXIT_FAILURE;
  }
   return EXIT_SUCCESS;
}
```


Fonctions de base (7)

- **Ecriture**
  - **un caractère**
    - **int fputc (int car, FILE* pFile);**
      - Écrit le caractère *car* dans le fichier associé à *pFile*.
      - Renvoie EOF en cas d'erreur ou 0 sinon.
      - **int putchar (int)** équivalent à **fputc** sur **stdout**.
  - **une chaîne de caractères**
    - **int fputs (char *pChaine, FILE* pFile);**
      - Écrit la chaîne *pChaine* dans le fichier associé à *pFile*.
      - Le caractère nul de fin de chaîne n'est pas écrit.
      - Renvoie EOF en cas d'erreur ou 0 sinon.

Exemple – fputc et fputs

fputc-s-test.c

```
#define POSIX_SOURCE 1
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>

int main (int argc, char ** argv) {
 FILE *ptEcr;

 if ( (ptEcr = fopen (argv[1], "w+")) == NULL) {
  perror ("fopen"); exit (1);
 }

 return (EXIT_SUCCESS);
}
```

```
/* écriture d'un caractère */
   if ((fputc('a',ptEcr))== EOF) {
      perror ("fputc");
      exit (1);
   }

  /* ecriture d'une chaîne */
   if (fputs ("bcd", ptEcr) == EOF) {
      perror ("fputs");
      exit (1);
   }

  fclose (ptEcr);
 return (EXIT_SUCCESS);
}
```


Fonctions de base (8)

- **Ecriture (cont)**
  - **Ecriture d'un tableau d'objets**

    size_t *fwrite (void *p, size_t taille, size_t nElems, FILE* pFile);
    - Écrit *nElems* objets de taille *taille* à partir de la *position courante* dans *pFile*
    - Le tableau d'objets à écrire est à l'adresse *p*.
    - Retourne le nombre d'objets écrits
      une valeur inférieure à *nELems* en cas d'erreur.

Fonctions de base (9)

- **Ecriture (cont)**
  - **Ecriture formatée**
    - **int printf (const char *format, ….);**
    - **int fprintf (FILE* pFile, const char *format, ….);**
      - Ecrit dans un fichier associé à *pFile* les valeurs des arguments converties selon le format en chaînes de caractères imprimables.
      - **printf** équivaut à **fprintf** sur **stdout**.
      - Retourne le nombre de caractères écrits ou un nombre négatif en cas d'erreur.

Exemple – fgets et fputs (fscp)

Copier le fichier argv[1] vers argv[2]

```
#define _POSIX_SOURCE 1

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

#define TAILLE_TAMPON 100

 FILE *fd1, *fd2;
 int nombre_car;
char tampon [TAILLE_TAMPON];

int main (int argc, char* argv []) {
  fd1 = fopen (argv[1],"r");
  fd2 = fopen (argv[2], "w");

  if ( (fd1== NULL) || (fd2 == NULL)) {
    fprintf (stderr, "erreur fopen");
    return EXIT_FAILURE;
  }
  while (fgets (tampon, TAILLE_TAMPON, fd1) !=NULL)
    if ( fputs (tampon, fd2) == EOF){
      fprintf (stderr, "erreur fwrite\n");
      return EXIT_FAILURE;
    }
  fclose (fd1);
  fclose (fd2);

  if (ferror (fd1) ) {
    fprintf (stderr, "erreur lecture \n");
    return EXIT_FAILURE;
  }
  else
    return EXIT_SUCCESS;
  }
```


Exemple: fread et fwrite

Copier le fichier argv[1] vers argv[2]

```
#define _POSIX_SOUCE 1

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

#define TAILLE_TAMPON 100

FILE *fd1, *fd2;
 int nombre_car;
char tampon [TAILLE_TAMPON];

int main (int argc, char* argv []) {
  fd1 = fopen (argv[1],"r");
  fd2 = fopen (argv[2], "w");

  if ( (fd1== NULL) || (fd2 == NULL)) {
    fprintf (stderr, "erreur fopen");
    return EXIT_FAILURE;
  }
  while ((nombre_car = fread (tampon,   sizeof(char),
                TAILLE_TAMPON, fd1)) >0)
if ( fwrite (tampon, sizeof(char),nombre_car, fd2) !=
                nombre_car) {
      fprintf (stderr, "erreur fwrite\n");
      return EXIT_FAILURE;
}

  fclose (fd1);
  fclose (fd2);

if (ferror (fd1) ) {
    fprintf (stderr, "erreur lecture \n");
   return EXIT_FAILURE;
}
else
  return EXIT_SUCCESS;
 }
```


Fonctions de base (10)

- **Manipulation de la position courante**
  - **int fseek (FILE *pFile, long pos, int origine);**
    - positionne le curseur associé à *pFile* à la position *pos* relative à *origine*
    - origine: SEEK_SET, SEEK_CUR, SEEK_END
    - retourne une valeur non nulle en cas d'échec, 0 sinon
  - **void rewind (FILE *pFile);**
    - est équivalent à **fseek (pFile, 0L, SEEK_SET);**
  - **long ftell (FILE *pFile);**
    - retourne la position courante associée à *pFile*
      -1 en cas d 'erreur