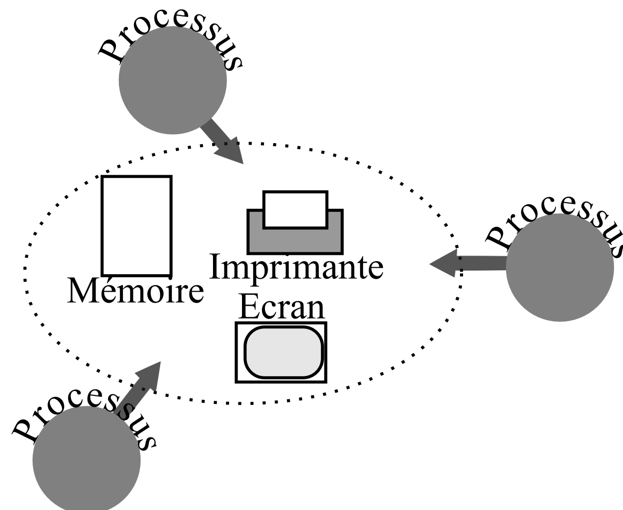
Processus et Fichiers

© Telecom-ParisTech BCI Informatique

Accès aux ressources

- Les processus peuvent avoir accès à une grande variété de ressources:



- Pour accéder aux différentes ressources dont il a besoin, l'utilisateur en donne le **nom,** il n'a pas besoin de connaître leur localisation matérielle, par exemple :

  - Accés à la mémoire : nom de variable,
    - `i = 2;`
  - Accès aux fichiers : nom du fichier :
    - `cat exo1.c, gcc –Wall exo1.c, ls > ls.out`
  - Accès aux périphériques : nom du périphérique:
    - `ls > /dev/pts/3, ssh dupont@champagne.enst.fr`

© Telecom-ParisTech BCI Informatique

Rôle du SE

- Le système masque la localisation matérielle et le fonctionnement des ressources, il les virtualise :

Ressource	Localisation matérielle
variable, objet	Adresse en mémoire (mécanisme d'adressage masqué)
fichier	Adresses de blocs sur un support permanent (accès DMA masqué).
périphérique	Adresse de l'UE et de ses registres sur le bus (architectures des bus masquée)

- Pour l'utilisateur les ressources (hormis la mémoire) sont vues comme des fichiers, il accède aux ressources en utilisant :
  - des commandes (Unix : `ls, cat`, ..., Windows : `DIR, TYPE, …`)
  - des méthodes (Java) ou fonctions (C) fournies par les langages de programmation,

© Telecom-ParisTech BCI Informatique

API pour les accès aux ressources

- Pour faire des opérations sur les fichiers :
    1. Il faut ouvrir le fichier en donnant son nom :
        - Fonctions `open` et `fopen` en C,
        - `FileInputStream`, `FileReader`, etc, en Java
    2. Le système associe alors un **descripteur** au nom du fichier. Ce descripteur peut être un simple numéro. Pour faire des opérations sur le fichier, le processus utilisera ce descripteur.
        - Le passage par un descripteur permet en particulier:
            - plusieurs ouvertures simultanées sur le même fichier,
            - la redirection des entrées-sorties (exemple : `ls>ls.dat`)

© Telecom-ParisTech BCI Informatique

E/S en C sous UNIX

- On peut utiliser deux bibliothèques :

  - celle dite de bas niveau (famille d'appels liés à `open` : `read`, `write`, etc)

  - celle dite de haut niveau (famille d'appels liés à `fopen` : `fread`, `fwrite`, etc)

- Relation entre processus et fichier : on va associer une variable locale au fichier dont on connait le nom global :

  - **Haut niveau :** pointeur sur une structure décrivant le fichier (variable de type `FILE*` renvoyée par `fopen`)

  - **Bas niveau :** descripteur de fichier : c'est à dire un index dans la table des fichiers ouverts par le processus (variable de type `int`, renvoyée par `open`)

- Par la suite on présente la bibliothèque de bas niveau (`open`, `read`, `write`, etc), les fonctions de haut niveau seront étudiées en annexe

© Telecom-ParisTech BCI Informatique

Rôle du système : La gestion des fichiers par Unix(1/2)

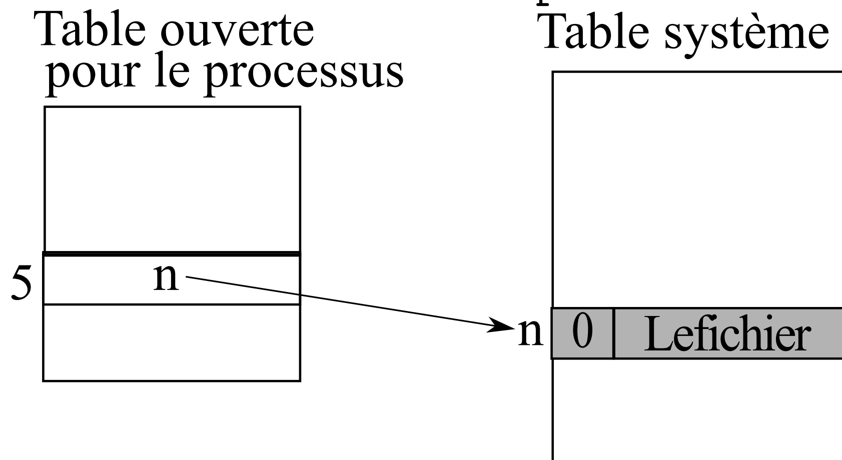
- Sous UNIX, le mécanisme d'accès aux fichiers repose sur l'utilisation de tables :
  1. Une table centrale et unique (System Open File Table), vue par tous les processus. Elle contient une entrée par fichier ouvert :
     - Chaque ouverture de fichier ajoute une entrée dans cette table.
     - Chaque entrée de cette table contient le nom du fichier et la valeur courante du pointeur dans celui-ci.
  2. Une table par processus. Ces tables contiennent une entrée par fichier ouvert **pour** le processus (pas forcément ouvert **par** lui).
     - Chaque entrée de cette table contient un pointeur vers une entrée de la table centrale.

© Telecom-ParisTech BCI Informatique

Rôle du système : La gestion des fichiers par Unix(2/2)

```
int i;
…
i = open("Lefichier", …);
```

- Effet de cet appel à `open` :
  1- création d'un fichier dont le nom est `LeFichier`,
  2- attribution à ce fichier d'une entrée dans la table générale (table système), ici son index est n. Le champ « pointeur » y est initialisé à 0.
  3- attribution d'une entrée dans la table associée au processus (la première entrée libre), ici son index est 5,
  4- cet index est la valeur de retour de `open` :



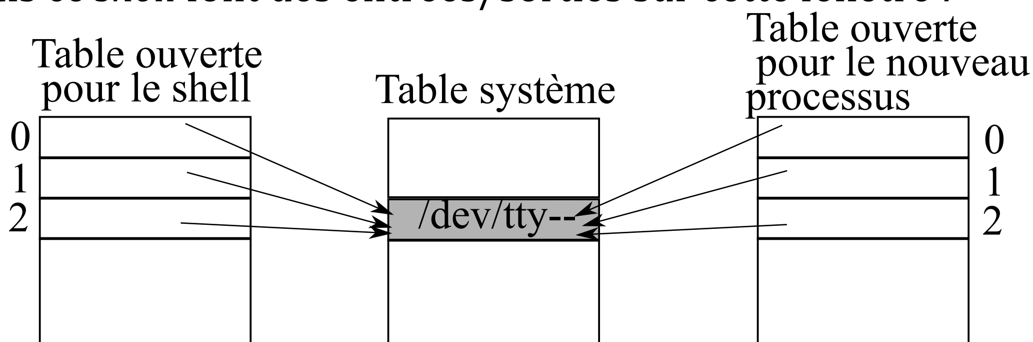
© Telecom-ParisTech BCI Informatique

Création de la table pour un processus

- Un processus hérite (par duplication) de la table du processus père,
- Donc, quand un processus est créé à partir du *shell*, il hérite d'une copie de la table du *shell*,
- Dans cette dernière, les premières entrées ont été affectées ainsi :

0	Entrée standard (`stdin`)
1	Sortie standard (`stdout`)
2	Sortie pour les messages d'erreur (`stderr`)

- Ces trois entrées pointent vers la **même** entrée de la table système. Cette entrée contient le nom de la fenêtre dans laquelle s'exécute le *shell*. C'est la raison pour laquelle toutes les commandes émises depuis ce *shell* font des entrées/sorties sur cette fenêtre :



- Nous allons maintenant illustrer la redirection des E/S dans le cas des commandes lancées depuis le *shell.*

© Telecom-ParisTech BCI Informatique

Redirection des Entrées/Sorties (1/2)

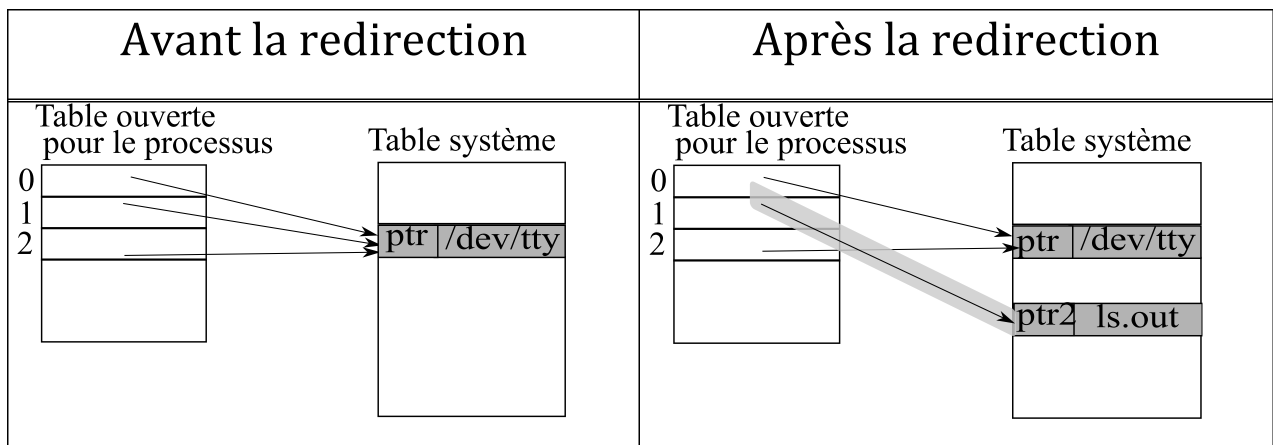
- La redirection des E/S montre l'intérêt de passer par un descripteur plutôt que de donner directement le nom de la ressource avec laquelle on travaille,
- Rappel : on peut rediriger les E/S en utilisant les symboles > et `< :`
  - \> redirige 1, la sortie standard,
  - < redirige 0, l'entrée standard
- Exemple de redirection :
  - `ls –l > ls.out :`

    On demande au shell de créer un processus dont les sorties ne se feront pas sur le terminal courant mais dans le fichier `ls.out` ,
  - le schéma de la page suivante indique comment se fait une redirection

© Telecom-ParisTech BCI Informatique

Redirection des Entrées/Sorties (2/2)

- Effet de la commande `ls –l > ls.out` au niveau des tables:
  1- création d'un fichier dont le nom est `ls.out`,
  2- attribution à ce fichier d'une entrée dans la table générale des fichiers ouverts,
  3- modification du contenu de l'entrée 1 de la table du processus shell. Cette entrée pointe maintenant vers l'entrée attribuée à `ls.out` dans la table générale
- Schéma de fonctionnement :



© Telecom-ParisTech BCI Informatique

On obtient le nom du terminal courant en utilisant la commande `tty.`
Par défaut 0, 1 et 2 sont associés au terminal courant (/dev/tty).
Chaque open sur un fichier renvoie le numéro de l'entrée qui a été associée à ce fichier dans la table des fichiers ouverts par le processus.
L'allocation se fait ainsi : la table est scrutée à partir de l'entrée zéro, et **la première entrée libre** est associée au fichier à ouvrir.

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
int main(void) {
    int Ret_open, i;

    for (i=0; i<3; i++){
    Ret_open = open("Mon_fichier", O_RDONLY);
    printf("Retour de open = %d\n", Ret_open);
    }
    close(0);

    for (i=0; i<3; i++){
    Ret_open = open("Mon_fichier", O_RDONLY);
    printf("Retour de open = %d\n", Ret_open);
    }
    return 1;
}
/****** resultat:
 Retour de open = 3
 Retour de open = 4
 Retour de open = 5
 Retour de open = 0
 Retour de open = 6
 Retour de open = 7
*******/
```

Commentaire:
L'appel à `close` a libéré l'entré d'index 0, qui devient ainsi la première entrée libre.

Exemple d'exécution de la commande `limit` :

```
cputime         unlimited
filesize        unlimited
datasize        6MB
stacksize       8MB
coredumpsize    0kB
memoryuse       unlimited
memorylocked    unlimited
maxproc         100
descriptors   256     (256 fichiers maximum ouverts simultanémént)
```

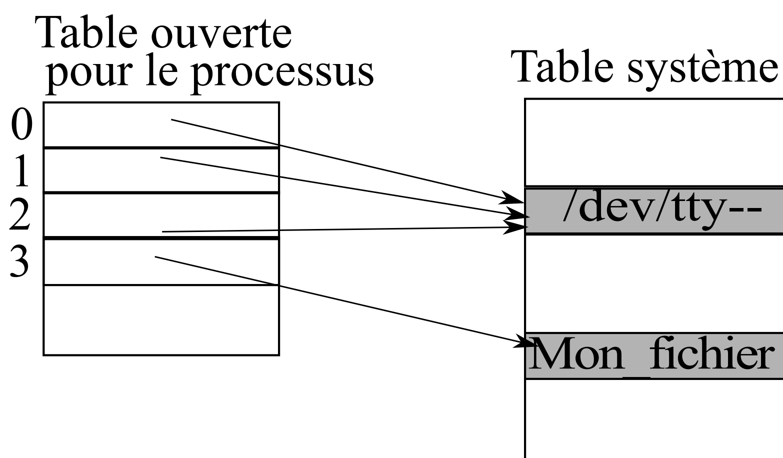

© Telecom-ParisTech BCI Informatique

Attribution des entrées dans les tables

- Lorsqu'un fichier est ouvert, le système lui associe la première entrée **libre** dans la table des fichiers ouverts pour le processus,
- Evolution de la table précédente après un premier appel à `open` :

Programme	Résulat

`int Ret_open;`
`Ret_open = open("Mon_fichier", O_RDONLY);`
`printf("retour de open = %d\n", Ret_open);`
	`retour de open = 3`



- Remarques*:*
  - `/dev/tty` a été ouvert **par** le shell : la table du processus contient tous les descripteurs vers les fichiers ouverts **pour** le processus (par héritage et par lui-même), la table système contient une entrée par **ouverture** de fichier.

© Telecom-ParisTech BCI Informatique

E/S bas niveau Ouverture ou creation

- Avant d'accéder à un fichier il faut l'ouvrir par un **open:**

```
int Ret_open;
•••
Ret_open = open("Fichier1", O_RDWR);
  if (Fichier == -1) {
      perror("open Fichier1");
      exit(1);
    }
```

- Pour créer un fichier: soit on utilise open avec les bons paramètres (exemples plus loin dans ce cours), soit on utilize creat:

```
int Ret_creat;
…
Ret_creat = creat("Fichier1", 0666);
```

- Rq : changer les droits d'accès :

```
int chmod(char* nom, int mode) ;
```


© Telecom-ParisTech BCI Informatique

Lecture/écriture dans un fichier E/S bas niveau

- Lecture, écriture et fermeture d'un fichier :

```
int write(int fildes, char* buffer, unsigned nbyte)
int  read(int fildes, char* buffer, unsigned nbyte)
int close(int fildes)
```

- Valeur de retour :

  - read/write : un entier positif correspondant au nombre d’octets lu/écrits. -1 en cas d’erreur.
  - close : 0 en cas de succès et -1 en cas d’echec.

© Telecom-ParisTech BCI Informatique

Exemple (open, read, write) :

```
int main(int argc, char *argv[]){
        int Mon_Fic, Ret_Read, Ret_Write;
        char Tab[512];
        /* ouverture en lecture*/
        Mon_Fic = open("toto", O_RDONLY);
        if(Mon_Fic == -1){
           perror("open");
           •••
        }
        while((Ret_Read = read(Mon_Fic, Tab, 512)) > 0){
           •••
           Ret_Write = write(1, Tab, Ret_Read);
           if(Ret_Write == -1){
              /* Traiter l'erreur */
           }
        }
        close(Mon_Fic) ;
}
```


© Telecom-ParisTech BCI Informatique

```
/**************************************
autre exemple, open, creat, read et
write ********************************/

•••
int main(int argc, char *argv[]){
  int i, f1, f2;
•••
  f1 = open(argv[1],0);
  if(f1 == -1){
    perror("open");
    exit (1);
    }
  f2 = creat(argv[2],0666);
  if(f2 == -1){
    perror("creat");
    exit (1);
    }

  while(1){
     i = read(f1,&c,1);
     if(i != 1) break;
     write(f2, &c, 1);
    }
}
•••
```


© Telecom-ParisTech BCI Informatique

E/S bas niveau
Positionnement dans un fichier

- Positionnement dans un fichier :

```
long lseek(int fildes, long offset, int from);
        from :
        0       depuis le début
        1       depuis la position courante
        2       à partir de la fin
```

Retourne :

- la position du pointeur à partir du début du fichier,
- -1 en cas d'erreur.

© Telecom-ParisTech BCI Informatique

Partage de fichiers

- Le partage de fichiers peut se faire de deux façons :

    1. en utilisant deux pointeurs différents pour parcourir le fichier partagé,

    2. en mettant en œuvre un pointeur unique,

- Pointeurs séparés :

  un ou plusieurs processus font accès au même fichier en utilisant des pointeurs différents

- Partage de pointeur :

  plusieurs processus font accès au même fichier en utilisant le même pointeur

© Telecom-ParisTech BCI Informatique

Partage de fichier : Pointeurs séparés (1/2)

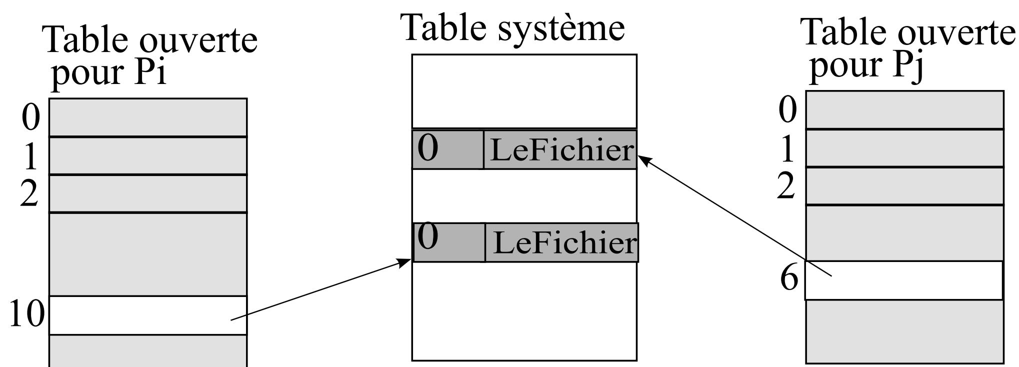
- On suppose ici que d'autres appels à `open` ont précédé ceux faits par Pi et Pj sur le fichier `LeFichier`:

Programme exécuté par Pi	Programme exécuté par Pj

`int main(void){`
`int Ret;`
`•••`
`Ret= open ("LeFichier", O_RDONLY);`
`printf ("retour de open = %d\n", Ret);`
`•••`
	`int main(void){`
	`int Ret;`
	`•••`
	`Ret= open ("LeFichier ", O_RDONLY);`
	`printf ("retour de open = %d\n", Ret);`
	`•••`

Résultat : `retour de open = 10`	Résultat : `retour de open = 6`

- Effet de ces deux appels à `open` sur les tables :



- Une entrée de la SOFT contient deux champs :
  1. la valeur du curseur dans le fichier (0 après une ouverture),
  2. nom du fichier complet (*pathname*,non représenté sur le schéma).

© Telecom-ParisTech BCI Informatique

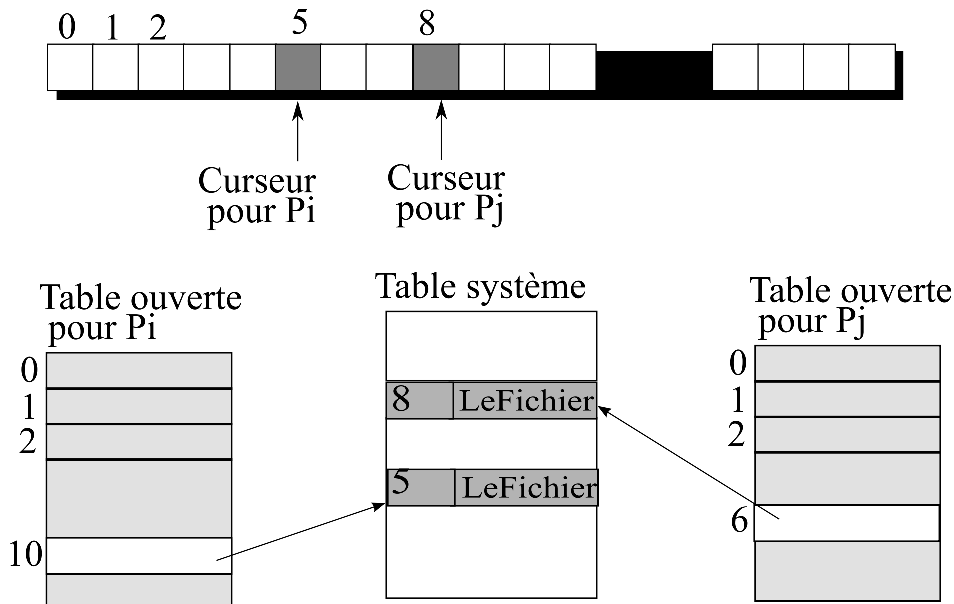
Partage de fichier : Pointeurs séparés (2/2)

- On suppose ici que d'autres appels à `open` ont précédé ceux faits par Pi et Pj déplacent chacun leur propre curseur dans le fichier partagé :

Instruction exécuté par Pi	Instruction exécuté par Pj

`/* Déplacer le curseur de 5 carcatères */`
`lseek``(Ret_open, 5, SEEK_CUR);`
`•••`
	`/* Déplacer le curseur de 8 carcatères */`
	`lseek``(Ret_open, 8, SEEK_CUR);`
	`•••`

- Chaque, processus déplace son curseur indépendamment de l'autre. Effet de ces deux appels à `lseek` sur les tables et dans le fichier partagé:



© Telecom-ParisTech BCI Informatique

Soit le programme :

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

int main (void){
    int Ret_open, Etat, Ret_seek, Ret_fork;

    Ret_open = open("Mon_fichier",0);

    /* Le pere positionne le curseur en 3 dans le fichier */
    lseek(Ret_open, 3, SEEK_CUR);

    /* Creation d un processus fils */
    Ret_fork = fork();

    if (Ret_fork == 0) {
    /* Le fils decale le curseur de 7 positions dans le fichier */
    Ret_seek = lseek(Ret_open, 7, SEEK_CUR);
    printf("Pid : %d : Valeur de Ret_open= %d et Ret_seek=  %d\n",
           (int)getpid(), Ret_open, Ret_seek);
    exit (1);
    }

    wait(&Etat);

    /* verifier la position courante  dans le fichier
     pour chaque processus */
    Ret_seek = lseek(Ret_open, 0, SEEK_CUR);
    printf("Pid : %d : Valeur de Ret_open= %d et Ret_seek=  %d\n",
       (int)getpid(), Ret_open, Ret_seek);
    return 1;
}

/****Resultat :
 Pid : 5779 : Valeur de Ret_open= 3 et Ret_seek=  10
 Pid : 5778 : Valeur de Ret_open= 3 et Ret_seek=  10
****/
```

On constate que les deux processus trouvent le curseur en 10 :

- Le fils a trouvé le curseur en 3 et l'a avancé de 7, il est donc en 10
- Le père le voit aussi en 10.

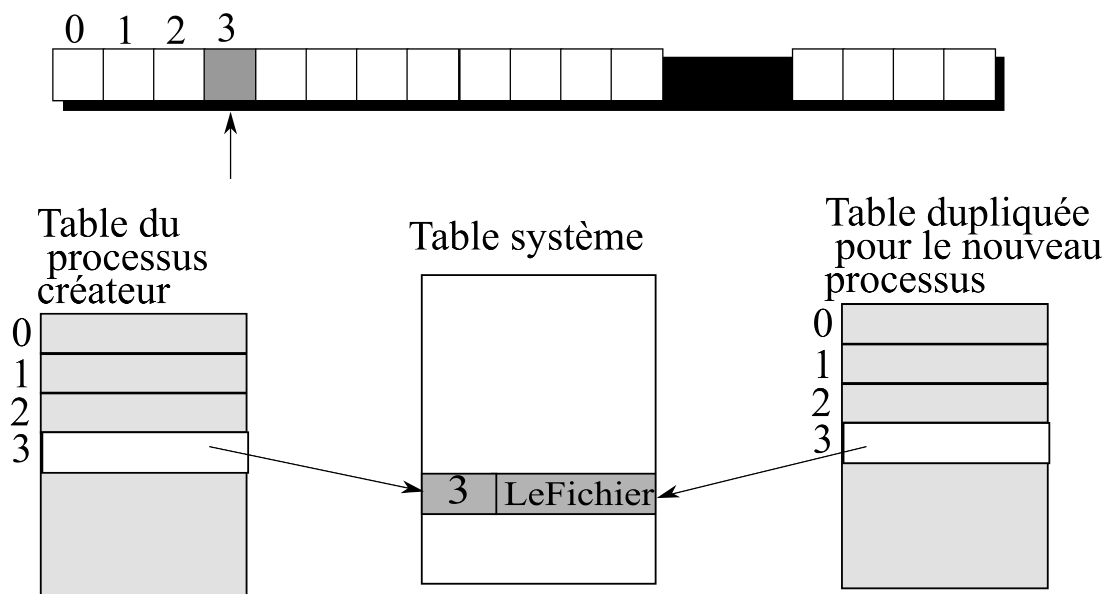
© Telecom-ParisTech BCI Informatique

Partage de fichier : Pointeur partagé (1/2)

- Soit cet extrait du programme :

```
Ret_open =  open("LeFichier", O_RDWR) ;

/* Positionner le curseur en 3 */
lseek(Ret_open, 3, SEEK_CUR);

/* Creation d un processus  fils */
Ret_fork = fork();
•••
if (Ret_fork == 0) {
       /* Decaler le curseur de 7 positions */
        lseek(Ret_open, 7, SEEK_CUR);
        •••
    }
•••
```

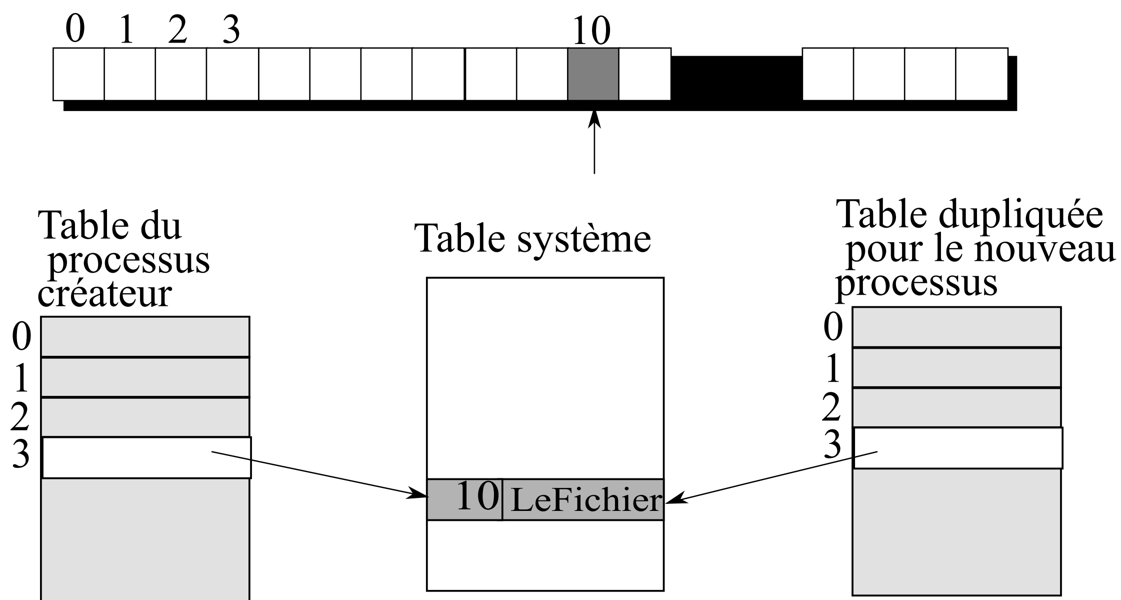
- Configuration des tables et position du curseur dans le fichier partagé après exécution de `fork` :



© Telecom-ParisTech BCI Informatique

Partage de fichier : Pointeur partagé (2/2)

- Configuration des tables et position du curseur dans le fichier partagé après exécution de `lseek(7)` par le processus fils :



© Telecom-ParisTech BCI Informatique

```
/*******************************************************
         synchro par lockf
 remarque : les processus  qui n'utilisent pas lockf
 accedent sans controle au fichier
*******************************************************/
•••
int  main (int argc, char *argv[]){
•••
if((Sortie = open (argv[1], O_RDWR)) == -1) ;
•••
/* verrouillage de tout le fichier */
lockf (Sortie, F_LOCK , 0);
printf ("Pid %d : entree dans SC\n", (int)getpid() );

/*----------------------------------------------
  Debut de section critique
  ----------------------------------------------*/
   •••
   write (Sortie, Tab, strlen(Tab));
   •••
/*----------------------------------------------
  Fin de section critique
  ----------------------------------------------*/
•••
/* se repositionner au debut du fichier , sinon on ne
  deverrouille que ce qui suit la position courante */
lseek(Sortie, 0, SEEK_SET);
lockf (Sortie,  F_ULOCK , 0);
      printf ("Pid %d :  sortie SC, =  %d\n", (int)getpid());
•••
}
```


© Telecom-ParisTech BCI Informatique

Synchronisation des accès Aux fichiers (1/2)

- Pour gérer la concurrence sur les fichiers partagés (**modèle lecteurs/écrivains**), Unix propose des verrous spécifiques qui sont mis en œuvre en utilisant la fonction `lockf`.
- Ce sont des « *advisory locks* » c'est à dire que le passage par un verrou n'est pas obligatoire pour accéder à un fichier partagé.
- Le verrouillage et le déverrouillage se font sur un nombre d'octets donné en partant de la position courante. Ceci évite le problème du **faux partage** : verrouiller toute la ressource, alors que seul un sous-ensemble est utilisé.

verrouillage : `lockf(fichier, F_LOCK, nb_octets)`
déverrouillage : `lockf(fichier, F_ULOCK, nb_octets)`
où `fichier` est la valeur renvoyée par un `open`.
si `nb_octets = 0`, tout le fichier est verrouillé à partir de la position courrante du pointeur.

- **Interblocage** possible, un message d'erreur est renvoyé.

- Remarque : `lockf` agit à partir de la position courante, utiliser `lseek` pour gérer cette position.

© Telecom-ParisTech BCI Informatique

```
/*-------------------------------------------------------
    utilisation d'un fichier-verrou
    ATTENTION, utiliser un fichier visible
    PAR TOUS LES UTILISATEURS du verrou
    cet exemple suppose deux processus
    dans le meme repertoire  !!!!!!!!!
    UTILISER LOCKF
    -------------------------------------------------------- */

#define VERROU "verrou"

 int main (void){
 int f2, i, ret;

 while (1){
  printf ("Pid %d : avant entree dans SC\n", (int)getpid() );

  do{
    f2=open (VERROU, O_EXCL | O_CREAT | O_TRUNC | O_EXCL , 0666);
    sleep (2);  /* pour limiter l'attente active */
    }
   while (f2  == -1) ;

   printf ("Pid %d : entree dans SC\n", (int)getpid() );
    /*----------------------------------------------
     Debut de section critique
     ----------------------------------------------*/
    for (i=0;  i<10000000; i++) ;
    /*----------------------------------------------
     Fin de section critique
     ----------------------------------------------*/
  ret = remove (VERROU);
  sleep(2);
  printf ("Pid %d : sortie SC, ret = %d \n", (int)getpid(), ret );
  }
return 1;
}
```

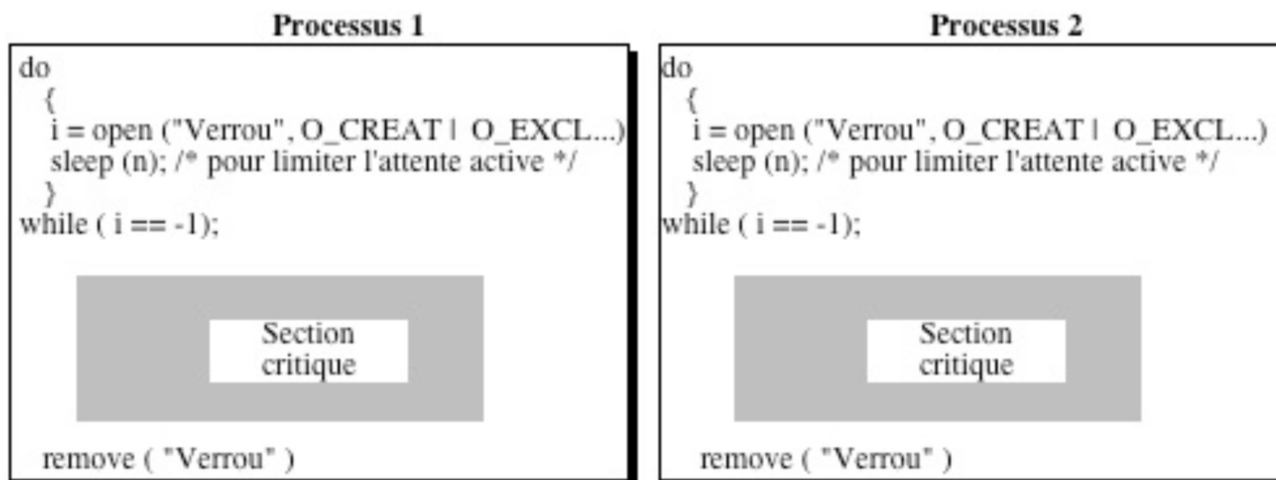

© Telecom-ParisTech BCI Informatique

Synchronisation des accès Aux fichiers (2/2)

• Mauvaise mise en œuvre : utilisation de "fichiers-verrous" :

```
ret = open("Verrou", O_CREAT | O_EXCL,  0666)
```

on vérifie la valeur de retour de open pour savoir si le verrou existe :



• Conditions de bon fonctionnement (problème d'unicité du non du verrou et de sa localisation) :

- mettre le fichier-verrou dans `/tmp` ou tout autre répertoire accessible par **tous** les processus concernés,
- donner à ce fichier-verrou un nom bien particulier.

• Exemple : voir les verrous utilisés par le navigateur *firefox* dans : `.mozilla/firefox`

© Telecom-ParisTech BCI Informatique

Le modèle producteur/consommateur

- Les accès à un fichier déclaré comme « tube » (*pipe*) sont gérés par Unix suivant le schéma producteur/consommateur :
    - Les **read** et **write** sur ce fichier seront en fait respectivement des « **consommer »** et « **déposer » ,**
- Conséquences :
    - Les lectures sont destructrices : un caractère « lu » est retiré du tube,
    - Un read est bloquant si le tube est vide,
    - Un write est bloquant si le tube est plein.

- Remarque :
    - La synchronisation est assurée par le système

© Telecom-ParisTech BCI Informatique

Exemple de création d'un tube nommé, il sera accessible à tous les utilisateurs puisque les droits sont read-write pour toutes les catégories d'utilisateurs (0666).
Les **read** et **write** sur ce fichier seront en fait des **consommer** et **déposer**, c'est à dire que le read sera bloquant si le tube est vide et le write bloquant s'il est plein.

```
/* Creation d'un tube nomme visible de partout */
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>

int main(int nb_arg, char *argv[]){
  int Ret_mknod;
  if (nb_arg != 2){
    printf("Utilisation : %s Nom_du_tube a creer\n", argv[0]);
    exit(1);
    }

  Ret_mknod = mknod(argv[1], S_IFIFO | 0666, 0);
  if (ret_mknod != 0){
    perror("mknod");
    exit(2);
    }
  printf("On a cree le tube  %s\n", argv[1]);

 •••
}
```


© Telecom-ParisTech BCI Informatique

Visibilité et durée de vie des différents types de `pipe`

- Tube standard (*process persistent*) :
    - il n'est vu que par les descendants d'un ancêtre commun, plusieurs producteurs/consommateurs possibles sur le tube,
    - il est détruit une fois que tous les processus qui l'utilisent sont terminés,
    - créé par la fonction `pipe.`
- Tube dit « nommés » (*named pipe*), *system persistent* :
    - accessible pour peu qu'on connaisse son nom et qu'on ait les droits d'accès adéquats,
    - il n'est pas détruit quand le processus qui l'a créé se termine,
    - créé par `mknod ou mkfifo`.

© Telecom-ParisTech BCI Informatique

Exemple d'utilisation de la primitive pipe correspondant au schéma ci-contre.

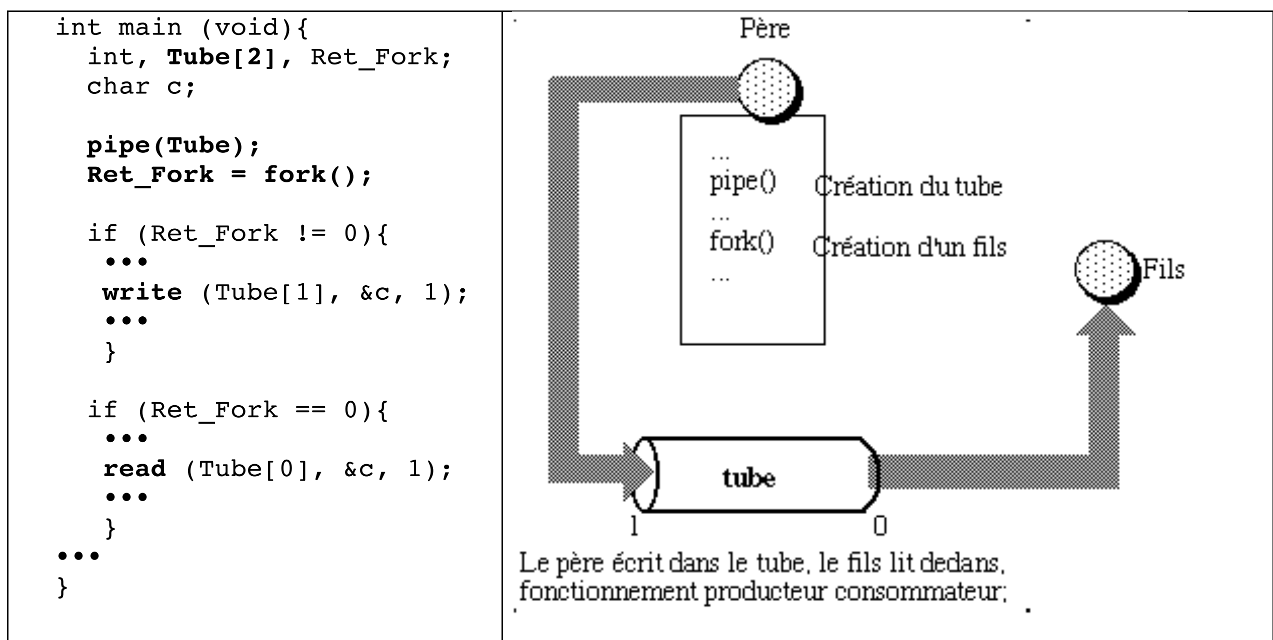
Le père dépose dans le tube les caractères saisis au clavier par un getchar. Le fils les affiche.
Les **read** et **write** sur ce fichier seront en fait des **consommer** et **déposer**, c'est à dire que le read sera bloquant si le tube est vide et le write bloquant s'il est plein.

```
•••
int  main (void){
  int Ret_Fork, Tube[2], Etat;
  char c;

  pipe(Tube);
  Ret_Fork = fork();

  if (Ret_Fork != 0 ){
•••
    close(Tube[0]);
    printf("Envoyer les caracteres!\n");
    while( (c = getchar()) != EOF )
      write (Tube[1], &c, 1);
    wait (&Etat);
    }

  if (Ret_Fork == 0 ){
      printf ("Fils pret en lecture.\n");
      close (Tube[1]);
      while ( read (Tube[0], &c, 1) )
      {
       printf ("caract recu = %c (0x%x)\n", c, c);
      }
     exit (0);
    }
•••
}
```


© Telecom-ParisTech BCI Informatique

Exemple d'utilisation

- Le programme suivant montre la création d'un schéma producteur/consommateur très simple par utilisation d'un fichier de type `pipe` en C :



```
int main (void){
  int, Tube[2], Ret_Fork;
  char c;

  pipe(Tube);
  Ret_Fork = fork();

  if (Ret_Fork != 0){
   •••
   write (Tube[1], &c, 1);
   •••
   }

  if (Ret_Fork == 0){
   •••
   read (Tube[0], &c, 1);
   •••
   }
•••
}
```


© Telecom-ParisTech BCI Informatique

Exemple d'utilisation en shell

- Le symbole **|** est un tube sur lequel arrivent les sorties du processus situé à sa gauche et où le processus situé à sa droite prend ses entrées :
- Exemples :
  1. chercher les processus créés par `dupont` :

```
$ps -aux | grep dupont
dupont  20826  0.0  0.0 148852  2272 ?      S  14:35 0:00 sshd:dupont@pts/0
dupont  20827  0.0  0.0 129008  4000 pts/0  Ss 14:35 0:00 -sh
dupont  20953  0.0  0.0 127172  2060 pts/0  S  14:36 0:00 bash
dupont  20997  0.0  0.0 122396  1232 pts/0  R+ 14:39 0:00 ps -aux
dupont  20998  0.0  0.0 111240   856 pts/0  S+ 14:39 0:00 grep dupont
```

  2. obtenir le nombre d'utilisateurs connectés sur une machine :

```
$who | wc –l
4
```

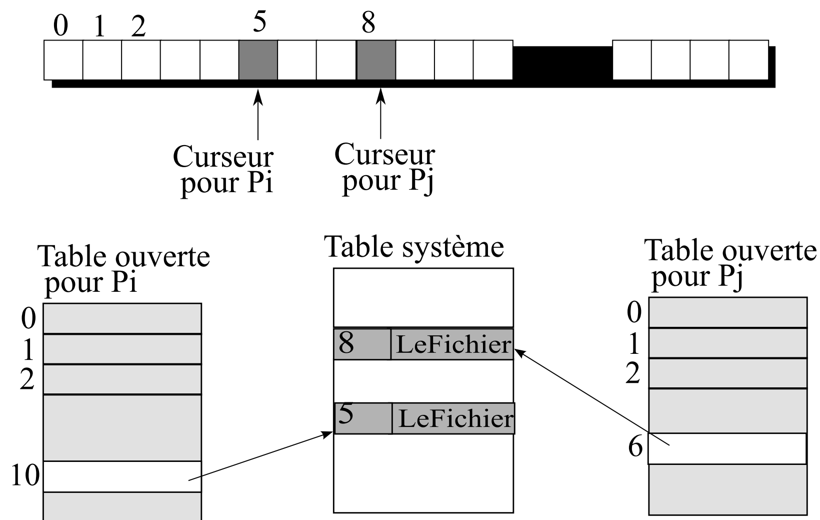
  3. trouver tous les fichiers de type `jpg` à partir du répertoire courant et les classer par ordre alphabétique :

```
$ ls -R | grep jpg | sort
arbo-1.jpg
arbo-2.jpg
dv.jpg
ecran-1.jpg
sw.jpg
```

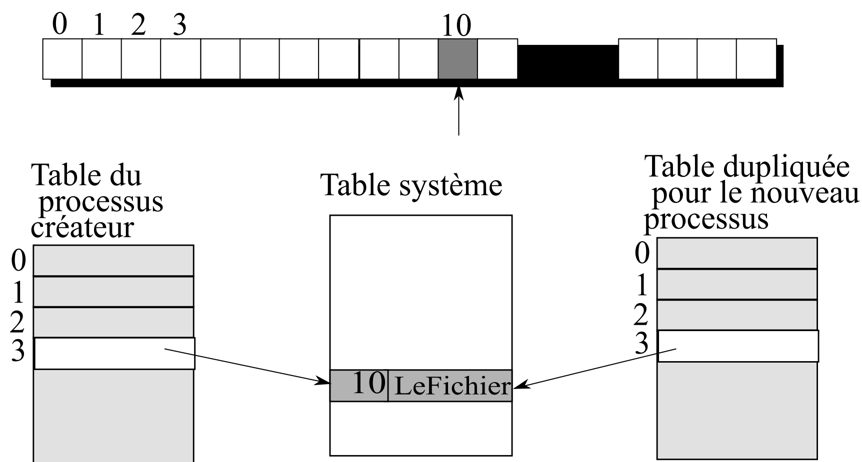

© Telecom-ParisTech BCI Informatique

Récapitulatif (1/2) : référencement

- Pointeurs séparés : un ou plusieurs processus font accès au fichier en utilisant des pointeurs différents



- Partage de pointeur : plusieurs processus font accès au fichier en utilisant le même pointeur



© Telecom-ParisTech BCI Informatique

Récapitulatif (2/2) : synchronisation

- Fichiers standards, ouverts par `open`, **modèle lecteurs/écrivains** :
  - les processus accèdent aux fichiers en fonction de leurs seuls droits d'accès, **pas de contrôle de concurrence**,
  - synchronisation en utilisant `lockf` ou `flock`.

- Fichiers tubes, **modèle producteur/consommateur :**
  - tube standard (ouvert par `pipe`) : il faut un ancêtre créateur
  - tube nommé (ouvert par `mkfifo` ou `mknod`) : il suffit d'en connaître le nom,
  - **synchronisation assurée** par le système,

© Telecom-ParisTech BCI Informatique

Annexes :

- Retour sur la bibliothèque d'entrées-sorties utilisée sur la partie du cours « langage C ».

© Telecom-ParisTech BCI Informatique

E/S haut niveau:

Note importante : Les fonctions de bas niveau sont spécifiques à UNIX. open renvoie une entrée dans la table des fichiers ouverts par le processus. Elle associe un numéro local à un nom global.

Ouverture et fermeture des fichiers avec la **biblio. de haut niveau** :

```
FILE * fopen (char * nom_externe, char * type)
```

"r"	lecture seule
"w"	écriture seule; s'il existe, il est détruit
"a"	concaténation; création s'il n'existe pas
"r+"	lecture et écriture (mise à jour)
"w+"	écriture et lecture
"a+"	lecture et concaténation

Exemple d'utilisation de `fopen`, on ouvre le fichier dont le nom est passé sur la ligne de commande :

```
#include <stdio.h>
int main(int argc, char *argv[]){

  FILE *Ptr_Fichier;
  •••
  Ptr_Fichier = fopen(argv[1],"r");
  if (Ptr_Fichier == (FILE *)0) {
    printf("Pas pu ouvrir %s !\n", argv[1]);
    exit(1);
    }
  printf("Fichier %s ouvert !\n", argv[1]);
  •••
}
```

Fermeture : `fclose(FILE *fp)`
Par exemple pour fermer l'entrée standard :

```
fclose(stdin);
```


© Telecom-ParisTech BCI Informatique

E/S haut niveau : formatées

	LECTURE	ECRITURE
clavier/écran	`Scanf`	`printf`
Fichier	`fscanf`	`fprintf`
Mémoire	`sscanf`	`sprintf`

- Syntaxe :

```
printf  (format, liste de valeurs)
fprintf (fichier, format, liste de valeurs)
sprintf (buffer, format, liste de valeurs)

scanf   (format, liste d'adresses)
fscanf  (fichier, format, liste d'adresses)
sscanf  (buffer, format, liste d'adresses)
```

- Exemples :

```
fprintf (stderr, "usage: %s [-d] [conf]", progname);
fprintf (stderr, "httpd: could not get socket\n");
sprintf (identite, "Client: %s@%s", user, machine);
```


© Telecom-ParisTech BCI Informatique

Format pour les appels de la famille printf :

Spécificateur	Correspondance
d	Entier
u	entier non signé
f	flottant
x	hexadecimal
c	caractère
s	chaîne de caractères
g	flottant double

Options :

l : complète le spécificateur pour les entiers longs- : justifier à gauche
+ : visualiser le signe (+ ou -)
0 : compléter par des 0

Formats variables :

printf ("%*.*s", 9, 5, chaine) ;

etc...(cf. man)

Formats pour les appels de la famille scanf :

%d	décimal
%x	hexadécimal
%s	lit une chaine jusqu'à blanc, TAB ou LF ajoute un '/n' à la chaine
%c	lit un caratère
%[..]	Si le caractère lu fait partie de l'ensemble, on continue la scrutation et l'affectation. Les autres caractères sont considérés comme des séparateurs. Dans le cas où le caractère '^' apparait en 1ère position (%[^..]), il s'agit du complément de cet ensemble.
%*d	Pas d'affectation à une variable pour ce format d'entrée
...	.cf le man ..

© Telecom-ParisTech BCI Informatique

E/S standards formatées : fscanf , exemple

• scanf retourne le nombre d'éléments affectés. Il s'arrête lors du premier conflit.

```
/*  lecture dans un fichier ou sur /dev/tty :
 on lit des mots et on s arrete sur EOF        */

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

#define MAX_MOTS 80

int main(int argc, char *argv[]){
  int ret_sc;
  FILE *Fichier;
  char Mots_lu[MAX_MOTS];

    Fichier = fopen(argv[1], "r");
  if(Fichier == NULL){
      printf("pb ouverture %s\n", argv[1]);
      return 1;
  }

  ret_sc=0;
  while ( (ret_sc != EOF) ){
      ret_sc=fscanf(Fichier,"%s", Mots_lu);
      if (ret_sc == 1) printf("mot lu : %s\n", Mots_lu);
  }

  printf("\n main : fin\n");
  return 0;
}
```


© Telecom-ParisTech BCI Informatique

E/S par caractère et par ligne

- Par caractère :

Lecture	Ecriture

`int getc(FILE *fp)`
`int fgetc(FILE *fp)`
`int getchar()` *raccourci pour: getc(stdin)*
 	
`int getw(FILE *fp)`
	`int putc(char c, FILE *pf)`
	`int fputc(c, pf)`
	`int putchar(c)` *raccourci pour: putc(c,stdout);*
	
	`int putchar(c)`

- Par ligne, par convention, une ligne de fichier texte se termine par `'\n'`.

Lecture	Ecriture
`char * gets(char *buffer)` *: depuis stdin* `char * fgets(char *buffer, int longmax, fp)`	`intputs(char *chaine)` *: vers stdout* `int fputs (char *chaine, fp)`

© Telecom-ParisTech BCI Informatique

Bibliothèque de haut niveau.
on lit ou écrit des enregistrements dont on donne la taille et le nombre :
Lecture :

```
int fread (char * buf, int size, int nb_rec, FILE * fp)
```

- Valeur retournée : nombre d'enregistrements de taille `size` lus.
- En cas d'erreur ou fin de fichier renvoie 0

Ecriture :

```
int fwrite (char * buf, int size, int nb_rec, FILE * pf)
```

- Valeur retournée : nombre d'enregistrements de taille `size` écrits.

© Telecom-ParisTech BCI Informatique

Informations sur un fichier :

```
void  stat(char* path, struct stat* buffer) ;
void fstat(int fildes, struct stat* buffer) ;
int access(char* path, int access_mode) ;
```

Bibliothèque de haut niveau (équivalent de lseek)

```
long fseek(FILE *pf, long offset, int from)
```

valeur de from	0	1	2
déplacement depuis	début	position courante	fin

`offset` est un déplacement en octets. Il peut être positif ou négatif.
`fseek` ne s'applique pas aux fichiers "`pipes`".

-Retourne 0 si ok, sinon une valeur ≠0 (!!!)

Pour obtenir la POSITION COURANTE :
`long ftell(FILE *pf)`
- retourne -1 si erreur.

Fichiers standards (associés au terminal courant)

	Haut niveau	Bas niveau	Défaut
Entrée standard	`stdin`	0	clavier
Sortie standard	`stdout`	1	écran
Sortie erreur	`stderr`	2	écran

Ces fichiers standards sont toujours ouverts, sauf si l'utilisateur les ferme explicitement.

© Telecom-ParisTech BCI Informatique

Exemple d'utilisation de fgets sur stdin :

On lit des chaines de caractères à l'aide de la fonction fgets. On s'arrête après avoir lu le mot « fin ».

```
while (fgets(Tab, sizeof(Tab), stdin) )
   {
     if (!strncmp(Tab, "fin", 3)) break;
     if (write(Sortie, Tab, strlen(Tab)) < 0)
       {
         printf("Erreur d'ecriture\n");
         break;
       }
```


© Telecom-ParisTech BCI Informatique