I3 – Algorithmique et programmation
Introduction à la programmation en langage C
Cours n°8
Manipulations de fichiers

Camille Coti
camille.coti@iutv.univ-paris13.fr

IUT de Villetaneuse, département R&T

2011 – 2012

1. Introduction à la manipulation de fichiers
   - Les fichiers sous Unix
   - Opérations sur les fichiers
2. Les fichiers en C
   - Descripteur de fichier
   - Ouverture et fermeture
3. Écriture
   - Écriture en mode caractères
   - Écriture binaire
   - Vidage des tampons d'écriture
4. Lecture
   - Lecture séquentielle
   - Lecture en mode caractères
   - Lecture binaire
   - Fin d'un fichier
5. Exemple : copie d'un fichier dans un autre
6. Synthèse des fonctions

Rappel

Liste des fichiers d'un répertoire sous Unix

```
coti@abidjan:~$ ls /
bin         initrd          mnt         sys         var
boot        initrd.img      opt         target      vmlinuz
cdrom       initrd.img.old  proc        tempo       vmlinuz.old
dev         lib             root        tempo1
etc         local           sbin        tmp
export      lost+found      selinux     users
home        media           srv         usr
```

Informations étendues sur un fichier

```
coti@abidjan:~$ ls -l /boot/vmlinuz-3.0.0
-rw-r--r-- 1 root root 2540656 25 juil. 15:11 /boot/vmlinuz-3.0.0
```

```
coti@abidjan:~$ file /boot/config-3.0.0
/boot/config-3.0.0: ASCII English text
coti@abidjan:~$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.18,
stripped
```

Rappels sur les droits d'accès

Quels droits

Permissions différentes

- En lecture ("r")
- En écriture ("w")
- En exécution ("x")

À qui

3 groupes :

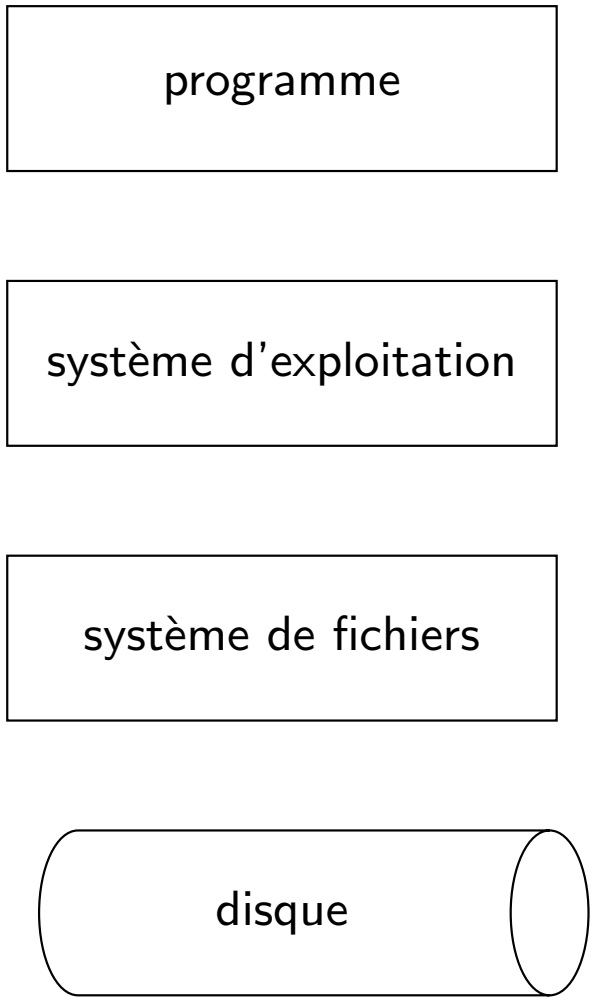
- Le propriétaire du fichier
- Un groupe d'utilisateurs
- Les autres utilisateurs

Les droits sont distincts pour ces trois groupes.

Représentation : 3 blocs de 3 lettres.
Exemple : `rwxr-x---`

- Le propriétaire peut lire, écrire, exécuter
- Le groupe peut lire et exécuter
- Les autres utilisateurs ne peuvent rien faire

Architecture



Le programme appelle des fonctions du système d'exploitation

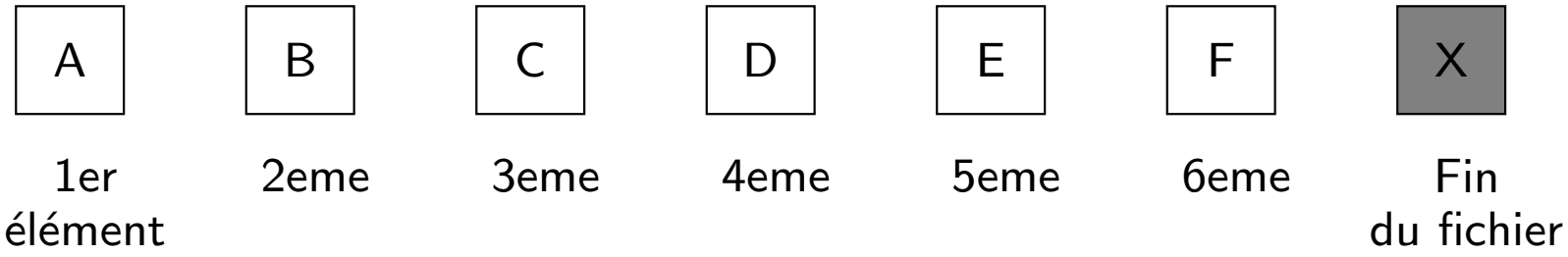
- Qui fait l'interface avec le système de fichiers
- Le système de fichier organise les données physiquement sur le disque

Exemples de systèmes de fichiers : ext3, ReiserFS, BtrFs, ZFS, tmpfs, HFS+ (Mac), NTFS (Windows)...

Fonctions d'entrées-sorties : `stdio.h`

Organisation des données

Les éléments d'un fichier sont rangés séquentiellement



Chaque fichier :

- A un **début**
- Se termine par une **marque de fin de fichier**
- Un curser marque la **position actuelle** dans le fichier
- On accède aux éléments de façon **séquentielle** ou **directe**

Opérations sur les fichiers

Opérations sur tous les fichiers

- Ouverture d'un fichier : on précise quel mode (lecture, écriture)
- Fermeture d'un fichier à la fin des opérations
- Test si on est à la fin du fichier

En écriture

- Création si il n'existe pas
- Écriture en fin de fichier
- Écriture en écrasant le fichier

En lecture

- Lecture séquentielle
- Accès direct

Attention aux droits d'accès !

- Vérifié par le système au moment de l'ouverture
- Exemple : ouverture d'un fichier en écriture $\rightarrow$ le système vérifie si on a les droits en écriture dessus

Descripteur de fichier

Le **type FILE***

- Défini dans `stdio.h`
- Pointeur de fichier → **descripteur de fichier**
- Manipulé pour les opérations dans les fichiers

```
1 #include <stdio.h>
2
3 /* declaration d'un descripteur de fichier */
4 FILE* fd;
```

Ouverture d'un fichier

Utilisation de la fonction **fopen()**

- Retourne de descripteur de fichier
- Prend en paramètres :
  - Le nom du fichier (`char*`)
  - Le mode d'ouverture (`char*`)

Modes d'ouverture :

- En lecture :
  - `r` : lecture à partir du début du fichier
- En écriture :
  - `w` : écriture à partir du début du fichier, en l'écrasant
  - `a` : écriture à partir de la fin du fichier, en le créant si il n'existe pas
- En lecture/écriture :
  - `r+` : lecture et écriture à partir du début du fichier
  - `w+` : lecture et écriture à partir du début du fichier, en l'écrasant
  - `a+` : lecture à partir du début du fichier et écriture à partir de la fin du fichier, en le créant si il n'existe pas

Le système vérifie si on a le droit d'ouvrir le fichier dans le mode demandé

Ouverture d'un fichier

En cas d'erreur : `fopen()` retourne `NULL`

- On teste la valeur de retour pour savoir si tout s'est bien passé

Exemple :

```
1  /* Declaration de deux descripteurs de fichier */
2  FILE* fd1;
3  FILE* fd2;
4
5  /* Ouverture d'un fichier en lecture */
6  fd1 = fopen( "/tmp/toto.txt", "r" );
7  if( NULL == fd1 ) { /* On teste le retour de fopen() */
8      printf( "Une erreur s\'est produite\n" );
9      return EXIT_FAILURE;
10 }
11
12 /* Ouverture d'un fichier en écriture */
13 fd2 = fopen( "/tmp/titi.txt", "w" );
14 if( NULL == fd2 ) { /* On teste le retour de fopen() */
15     printf( "Une erreur s\'est produite\n" );
16     return EXIT_FAILURE;
17 }
```

Fermeture d'un fichier

À la fin de l'utilisation d'un fichier, **on le ferme**

- Utilisation de la procédure **fclose()**
- Argument : un descripteur de fichier

Exemple :

```
1  /* Declaration d'un descripteur de fichier */
2  FILE* fd;
3
4  /* Ouverture d'un fichier en lecture */
5  fd = fopen( "/tmp/toto.txt", "r" );
6  if( NULL == fd ) { /* On teste le retour de fopen() */
7      printf( "Une erreur s\'est produite\n" );
8      return EXIT_FAILURE;
9  }
10
11 /* Fermeture du fichier */
12 fclose( fd );
```

Écriture formatée dans un fichier

Écriture **formatée**, en mode **caractère**

- Utilisation de la fonction **fprintf()**
- Proche de `printf()`, en passant le descripeur de fichier en 1er paramètre
- Retourne le nombre de caractères écrits (sans le \0 final)
- Fichier **texte**, contenant des caractères

Exemple :

```
1  /* Declaration d'un descripteur de fichier */
2  FILE* fd;
3  int ret;
4
5  /* Ouverture d'un fichier en ecriture */
6  fd = fopen( "/tmp/toto.txt", "w" );
7  if( NULL == fd ) { /* On teste le retour de fopen() */
8      printf( "Une erreur s\'est produite\n" );
9      return EXIT_FAILURE;
10 }
11
12 /* Écriture */
13 ret = fprintf( fd, "Toto %d\n", 5 );
14 printf( "%d caractères écrits\n", ret );
15
16 /* Fermeture du fichier */
17 fclose( fd );
```

Écriture caractère par caractère

Utilisation de la fonction **putc()**

- Écrit un caractère dans un fichier
- Prend en paramètres le caractère et le descripteur de fichier
- Retourne un entier : `EOF` si une erreur est survenue

Exemple :

```
1  /* Declaration d'un descripteur de fichier */
2  FILE* fd;
3  int ret;
4
5  /* Ouverture d'un fichier en écriture */
6  fd = fopen( "/tmp/toto.txt", "w" );
7  if( NULL == fd ) { /* On teste le retour de fopen() */
8      printf( "Une erreur s\'est produite\n" );
9      return EXIT_FAILURE;
10 }
11
12 /* Écriture */
13 ret = putc( 'a', fd ); /* 'a' est le caractère écrit */
14 if( EOF == ret ){ /* On teste le retour de putc() */
15     printf( "Une erreur s\'est produite\n" );
16     return EXIT_FAILURE;
17 }
18
19 /* Fermeture du fichier */
20 fclose( fd );
```

Écriture ligne par ligne

Définition d'une ligne :

- Chaîne de caractères se terminant par le caractère spécial \n

Utilisation de la fonction **fputs()**

- Arguments :
  - Un tableau de caractères terminé par \n
  - Le descripteur de fichier
- Retourne le nombre d'octets écrits, `EOF` si une erreur est survenue

```
1  FILE* fd;
2  int ret;
3  char* chaine = "coucou";
4  fd = fopen( "/tmp/toto.txt", "w" );
5  if( NULL == fd ) { /* On teste le retour de fopen() */
6      printf( "Une erreur s\'est produite\n" );
7      return EXIT_FAILURE;
8  }
9
10 ret = fputs( chaine, fd );
11 if( EOF == ret ){ /* On teste le retour */
12     printf( "Une erreur s\'est produite\n" );
13     return EXIT_FAILURE;
14 }
15
16 fclose( fd );
```

Écriture binaire dans un fichier

Écriture **binaire**, octet par octet

- En considérant un fichier comme une suite d'octets non interprêtés
- Utilisation de la fonction **fwrite()**
- Arguments :
  - Un pointeur vers le début du tampon mémoire à écrire dans le fichier
  - La taille de chaque élément à écrire (utilisation de `sizeof()`)
  - Le nombre d'éléments
  - Le descripteur de fichier
- Retourne le nombre d'éléments réellement écrits dans le fichier

Exemple :

```
1  FILE* fd;
2  int ret;
3  char* chaine = "coucou";
4  fd = fopen( "/tmp/toto.txt", "w" );
5  if( NULL == fd ) { /* On teste le retour de fopen() */
6      printf( "Une erreur s\'est produite\n" );
7      return EXIT_FAILURE;
8  }
9
10 ret = fwrite( chaine, sizeof( char), 6, fd );
11 printf( "%d elements ecrits\n", ret );
12
13 fclose( fd );
```

Vidage des tampons d'écriture

Les opérations sur disques sont **lentes**

- Optimisations du système d'exploitation
- Mise en tampon des opérations à effectuer
- Plusieurs écritures faites d'un seul coup

Conséquence : une écriture peut être effectuée en réalité plus tard sur le disque

- Difficulté de débuggage
- Si le programme plante : écriture pas forcément faite
- Si on lit par ailleurs le fichier en même temps : écriture peut-être retardée

Solution : vidage des tampons d'écriture

- Utilisation de la procédure **fflush()**
- Paramètre : descripteur de fichier
- Force l'écriture sur disque (ralentit le programme)

```
1 fflush( fd );
```

1 Introduction à la manipulation de fichiers
- Les fichiers sous Unix
- Opérations sur les fichiers

2 Les fichiers en C
- Descripteur de fichier
- Ouverture et fermeture

3 Écriture
- Écriture en mode caractères
- Écriture binaire
- Vidage des tampons d'écriture

4 Lecture
- Lecture séquentielle
- Lecture en mode caractères
- Lecture binaire
- Fin d'un fichier

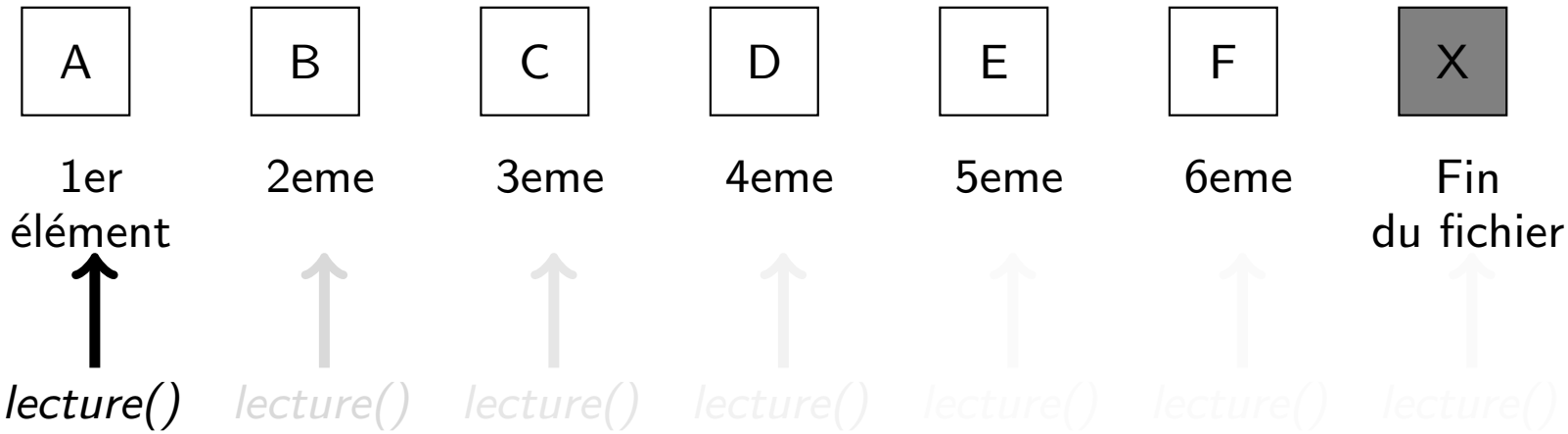
5 Exemple : copie d'un fichier dans un autre

6 Synthèse des fonctions

Lecture séquentielle

Lecture séquentielle = on lit les éléments les uns après les autres

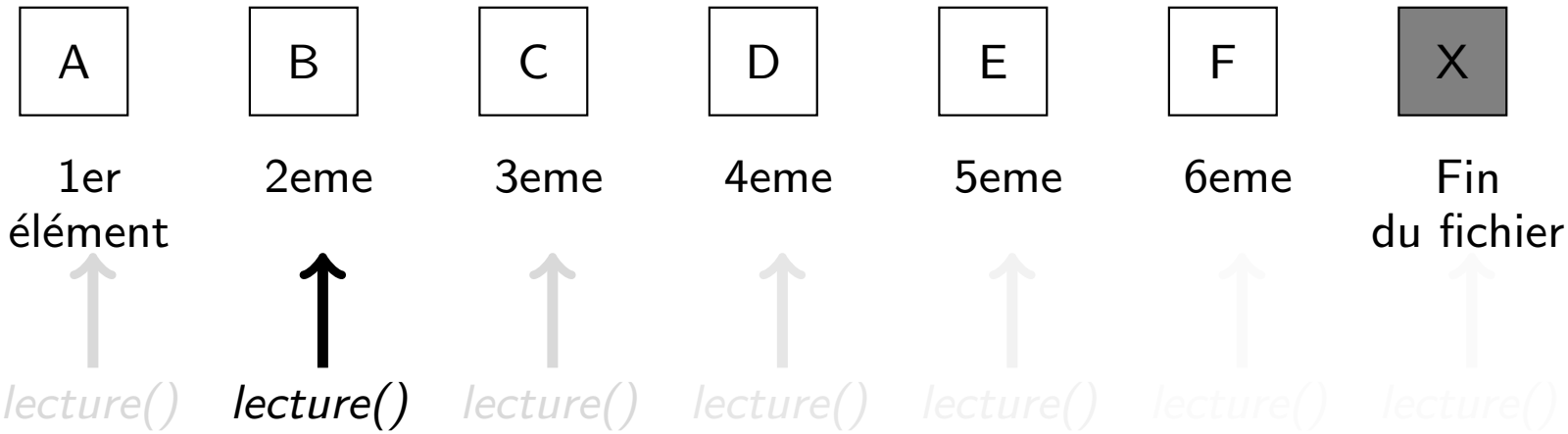
- On a un **marqueur (offset)** qui se déplace dans le fichier
- Quand on lit un élément (caractère, phrase, octet...) **l'offset est mis après** cet élément



Lecture séquentielle

Lecture séquentielle = on lit les éléments les uns après les autres

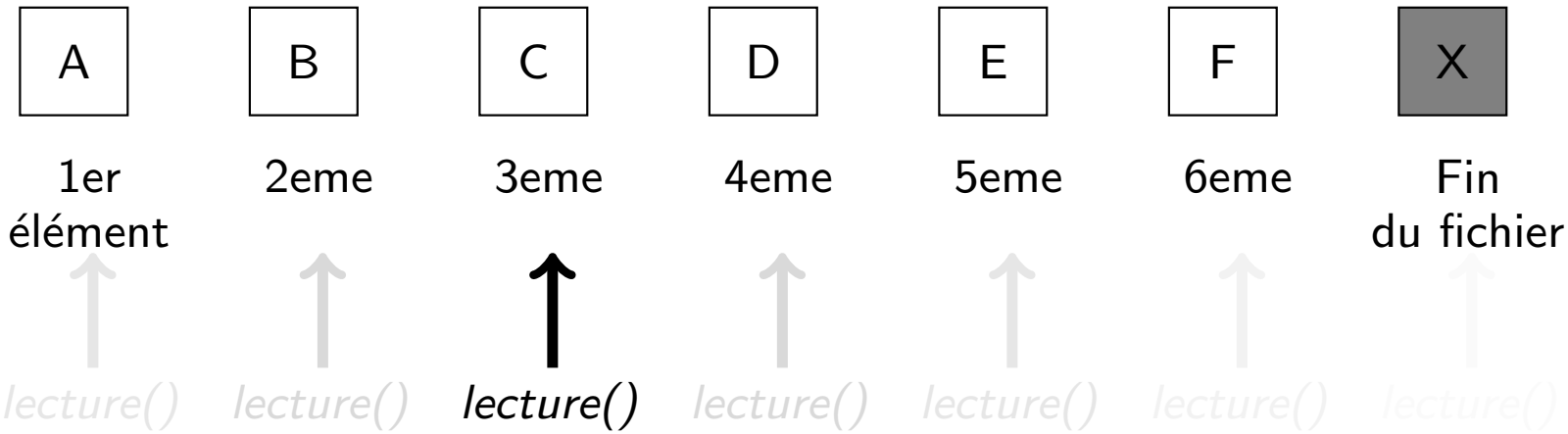
- On a un **marqueur (offset)** qui se déplace dans le fichier
- Quand on lit un élément (caractère, phrase, octet...) **l'offset est mis après** cet élément



Lecture séquentielle

Lecture séquentielle = on lit les éléments les uns après les autres

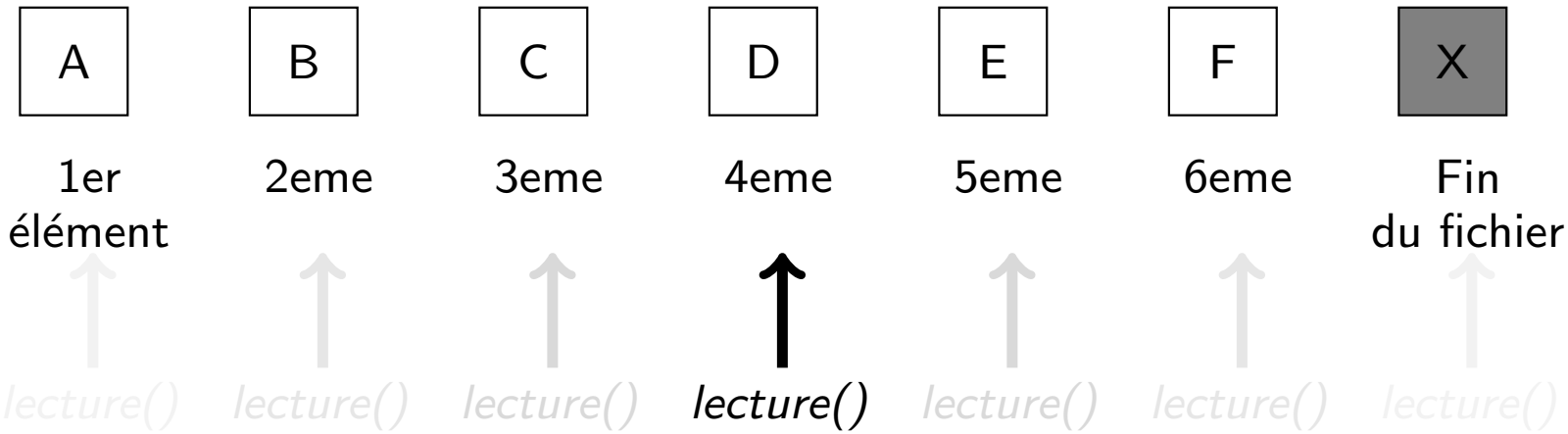
- On a un **marqueur (offset)** qui se déplace dans le fichier
- Quand on lit un élément (caractère, phrase, octet...) **l'offset est mis après** cet élément



Lecture séquentielle

Lecture séquentielle = on lit les éléments les uns après les autres

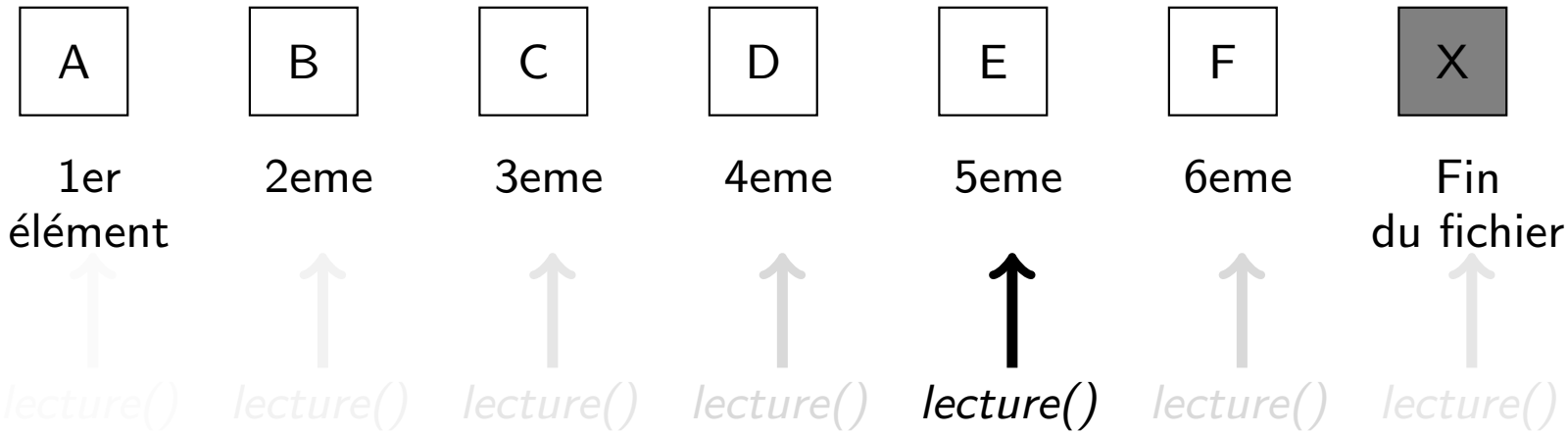
- On a un **marqueur (offset)** qui se déplace dans le fichier
- Quand on lit un élément (caractère, phrase, octet...) **l'offset est mis après** cet élément



Lecture séquentielle

Lecture séquentielle = on lit les éléments les uns après les autres

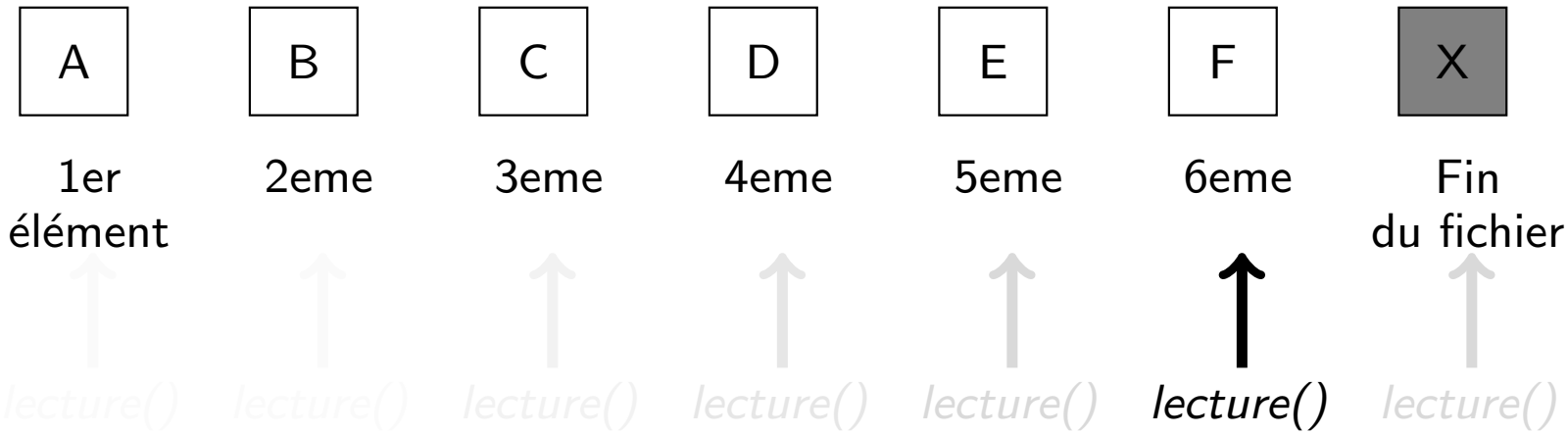
- On a un **marqueur (offset)** qui se déplace dans le fichier
- Quand on lit un élément (caractère, phrase, octet...) **l'offset est mis après** cet élément



Lecture séquentielle

Lecture séquentielle = on lit les éléments les uns après les autres

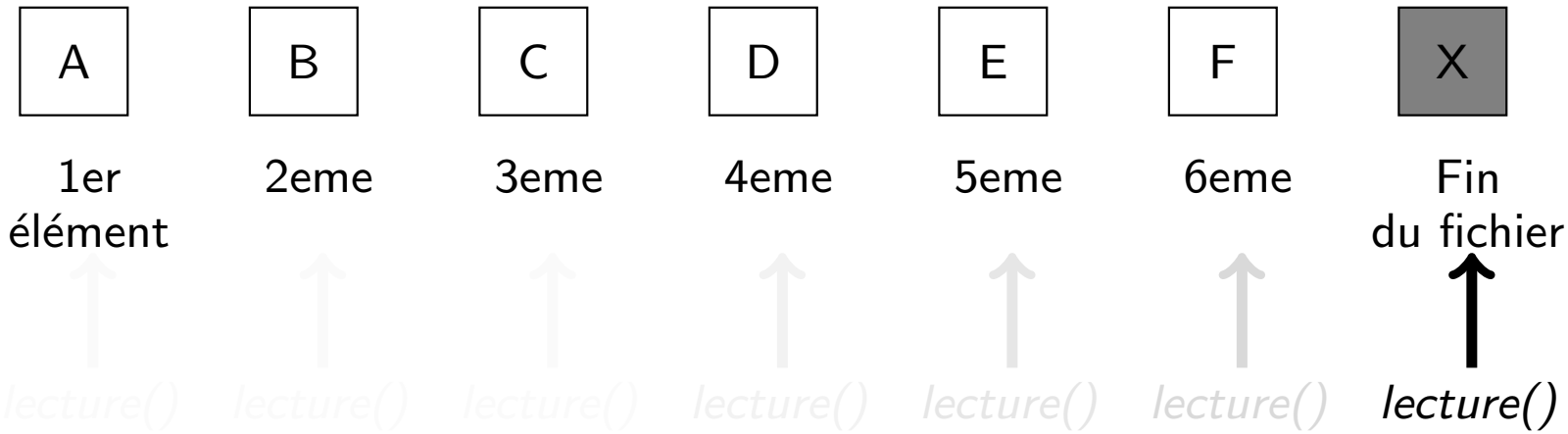
- On a un **marqueur (offset)** qui se déplace dans le fichier
- Quand on lit un élément (caractère, phrase, octet...) **l'offset est mis après** cet élément



Lecture séquentielle

Lecture séquentielle = on lit les éléments les uns après les autres

- On a un **marqueur (offset)** qui se déplace dans le fichier
- Quand on lit un élément (caractère, phrase, octet...) **l'offset est mis après** cet élément



Lecture formatée dans un fichier

Lecture **formatée**, en mode **caractère**

- Utilisation de la fonction **fscanf()**
- Proche de `fscanf()`, en passant le descripeur de fichier en 1er paramètre
- Retourne le nombre d'élements lus

Exemple :

```
1  /* Declaration d'un descripteur de fichier */
2  FILE* fd;
3  int ret, lu;
4
5  /* Ouverture d'un fichier en lecture */
6  fd = fopen( "/tmp/toto.txt", "r" );
7  if( NULL == fd ) { /* On teste le retour de fopen() */
8      printf( "Une erreur s\'est produite\n" );
9      return EXIT_FAILURE;
10 }
11
12 /* Lecture */
13 ret = fscanf( fd, "%d", &lu );
14 printf( "%d caractères lus : %d\n", ret, lu );
15
16 /* Fermeture du fichier */
17 fclose( fd );
```

Lecture caractère par caractère

Plusieurs possibilités

- Fonction **getc()** : lit un caractère depuis un descripteur de fichier
- Pas sure !
- Attention aux lectures et saisies : sources de failles de sécurité (débordements de tampons notamment)
- Utilisation de **fgetc()**
  - Argument : un descripteur de fichier
  - Retourne : un entier, qui contient le caractère lu, `EOF` si fin du fichier

Exemple :

```
1  FILE* fd;
2  int ret;
3  fd = fopen( "/tmp/toto.txt", "r" );
4  if( NULL == fd ) {
5     printf( "Une erreur s\'est produite\n" );
6     return EXIT_FAILURE;
7  }
8
9  do{
10    ret = fgetc( fd );  /* On lit le caractère suivant */
11    printf( "caractère lu %c\n", (char) ret );
12 } while( ret != EOF );
13
14 fclose( fd );
```

Lecture d'une ligne

Lecture d'une ligne d'au maximum un certain nombre de caractères

- Utilisation de la fonction **fgets()**
- Attention à **allouer l'espace mémoire** pour le tampon qui contient la ligne lue
- On précise le nombre maximum de caractères à lire : assez sûr car on ne lit pas plus que ce qui est demandé → limitation du risque de débordement de tampon
- Arguments :
  - Pointeur vers la chaîne de caractères qui va recevoir la ligne lue
  - Nombre maximum de caractères à lire
  - Descripteur de fichier
- Retourne un pointeur vers la chaîne de caractères lue (`NULL` si problème)

Lecture d'une ligne (suite)

Exemple :

```
1  FILE* fd;
2  int ret;
3  char tampon[LINEMAX];
4
5  fd = fopen( "/tmp/toto.txt", "r" );
6  if( NULL == fd ) {
7     printf( "Une erreur s\'est produite\n" );
8     return EXIT_FAILURE;
9  }
10
11 fgets( tampon, LINEMAX, fd );
12 if( NULL == tampon ) {
13    printf( "probleme lecture\n" );
14    fclose( fd );
15    return EXIT_FAILURE;
16 }
17 ret = strlen( tampon );
18 printf( "%d caracteres lus, %s\n", ret, tampon );
19
20 fclose( fd );
```

Lecture binaire d'un fichier

Fichier **binaire** lu octet par octet

- On lit des octets "bruts"
- Utilisation de la fonction **fread()**
- Arguments :
  - Un pointeur vers le début du tampon mémoire à écrire dans le fichier
  - La taille de chaque élément à lire
  - Le nombre maximum d'éléments à lire
  - Le descripteur de fichier
- Retourne le nombre d'éléments réellement lus dans le fichier
- Attention à allouer le tampon mémoire au préalable

Exemple :

```
1  FILE* fd;
2  int ret;
3  char tampon[LINEMAX];
4  fd = fopen( "/tmp/toto.txt", "r" );
5  if( NULL == fd ) {
6     printf( "Une erreur s\'est produite\n" );
7     return EXIT_FAILURE;
8  }
9  ret = fread( tampon, sizeof( int ), LINEMAX, fd );
10 printf( "%d caracteres lus, %s\n", ret, tampon );
11 fclose( fd );
```

Fin d'un fichier

Comment savoir qu'on a atteint la fin du fichier à lire ?

- **Retour des fonctions** de lecture
  - `EOF` pour `getc()`, `fgetc()`
  - 0 pour `fread()` (0 éléments lus)
  - `NULL` pour `fgets()`
- Fonction **`feof()`**
  - Paramètre : un descripteur de fichier
  - Retourne 0 si on n'est pas à la fin du fichier, autre chose si on y est

Exemple :

```
1  FILE* fd;
2  int ret;
3  fd = fopen( "/tmp/toto.txt", "r" );
4  if( NULL == fd ) {
5    printf( "Une erreur s\'est produite\n" );
6    return EXIT_FAILURE;
7  }
8
9  while( ! feof( fd ) ){
10   ret = fgetc( fd );
11   printf( "caractère lu %c\n", (char) ret );
12 }
13
14 fclose( fd );
```

Exemple : copie d'un fichier dans un autre

On veut copier un fichier dans un autre fichier

- On va donc **ouvrir 2 fichiers** :
  - Le fichier d'origine en **lecture**
  - Le fichier destination en **écriture**
- On lit le fichier d'origine et on écrit dans le fichier destination au fur et à mesure
- **Tant que** la fin du fichier d'origine n'est pas atteinte

Exemple d'implémentation : caractère par caractère

- On lit dans le fichier d'origine avec **fgetc()**
- On écrit avec le fichier destination ce qu'on a lu avec **putc()**
- On regarde si on a atteint la fin du fichier d'entrée avec **feof()**

Test des valeurs de retour

- Si `fopen()` retourne **NULL** : problème d'ouverture d'un fichier
- Si `putc()` retourne **EOF** : problème à l'écriture dans le fichier destination
- En cas de problème : on ferme les fichiers ouverts et on quitte

Exemple : copie d'un fichier dans un autre

```
1  FILE* fd_in;
2  FILE* fd_out;
3  int ret, lu;
4
5  /* Ouverture des fichiers*/
6  fd_in = fopen( "/tmp/toto.txt", "r" );
7  if( NULL == fd_in ) {
8    printf( "Une erreur s\'est produite\n" );
9    return EXIT_FAILURE;
10 }
11 fd_out = fopen( "/tmp/toto2.txt", "w" );
12 if( NULL == fd_out ) {
13   printf( "Une erreur s\'est produite\n" );
14   fclose( fd_in ); /* on ferme le fichier ouvert */
15   return EXIT_FAILURE;
16 }
17
18 while( ! feof( fd_in ) ) {
19   lu = fgetc( fd_in );              /* Lecture */
20   ret = putc( (char)lu, fd_out );   /* Écriture */
21   if( EOF == ret ) {
22     printf( "Une erreur s\'est produite\n" );
23     fclose( fd_in );
24     fclose( fd_out );
25     return EXIT_FAILURE;
26   }
27 }
28
29 /* Fermeture des fichiers */
30 fclose( fd_in );
31 fclose( fd_out );
```

Synthèse des fonctions

Ouverture / fermeture

- `fopen()` : ouvrir un fichier
- `fclose()` : fermer un fichier

Lecture / écriture

	Lecture	Écriture
Caractère	`fgetc()`	`putc()`
Ligne	`fgets()`	`fputs()`
Formatée	`fscanf()`	`fprintf()`
Binaire	`fread()`	`fwrite()`

Divers

- `fflush()` : écrire le contenu des tampons systèmes
- `feof()` : tester si on est à la fin du fichier