



Gestion des Entrées-Sorties

Présentation: Stéphane Lavirotte

Auteurs: ... et al*

**(*) Cours réalisé grâce aux documents de :
Jean-Paul Rigault**

Mail: Stephane.Lavirotte@unice.fr

Web: <http://stephane.lavirotte.com/>

Université de Nice - Sophia Antipolis



Gestion d'Entrées-Sorties

Accès aux périphériques



Terminologie

✓ Périphérique (« device »)

- Unité physique support d'E/S
- Disque, terminal, imprimante

✓ Norme d'interface des périphériques

- Multi-vendeurs
- SCSI, IDE (disques) ; RS232c, v24 (terminaux, imprimantes, modems...) ;
- IEEE 802.3 (Ethernet)
- IEEE 488 (instrumentation)

✓ Contrôleur

- Interface matérielle entre le système informatique et un ou plusieurs périphériques
- Dépend (de la norme d'interface) du périphérique

✓ Canal

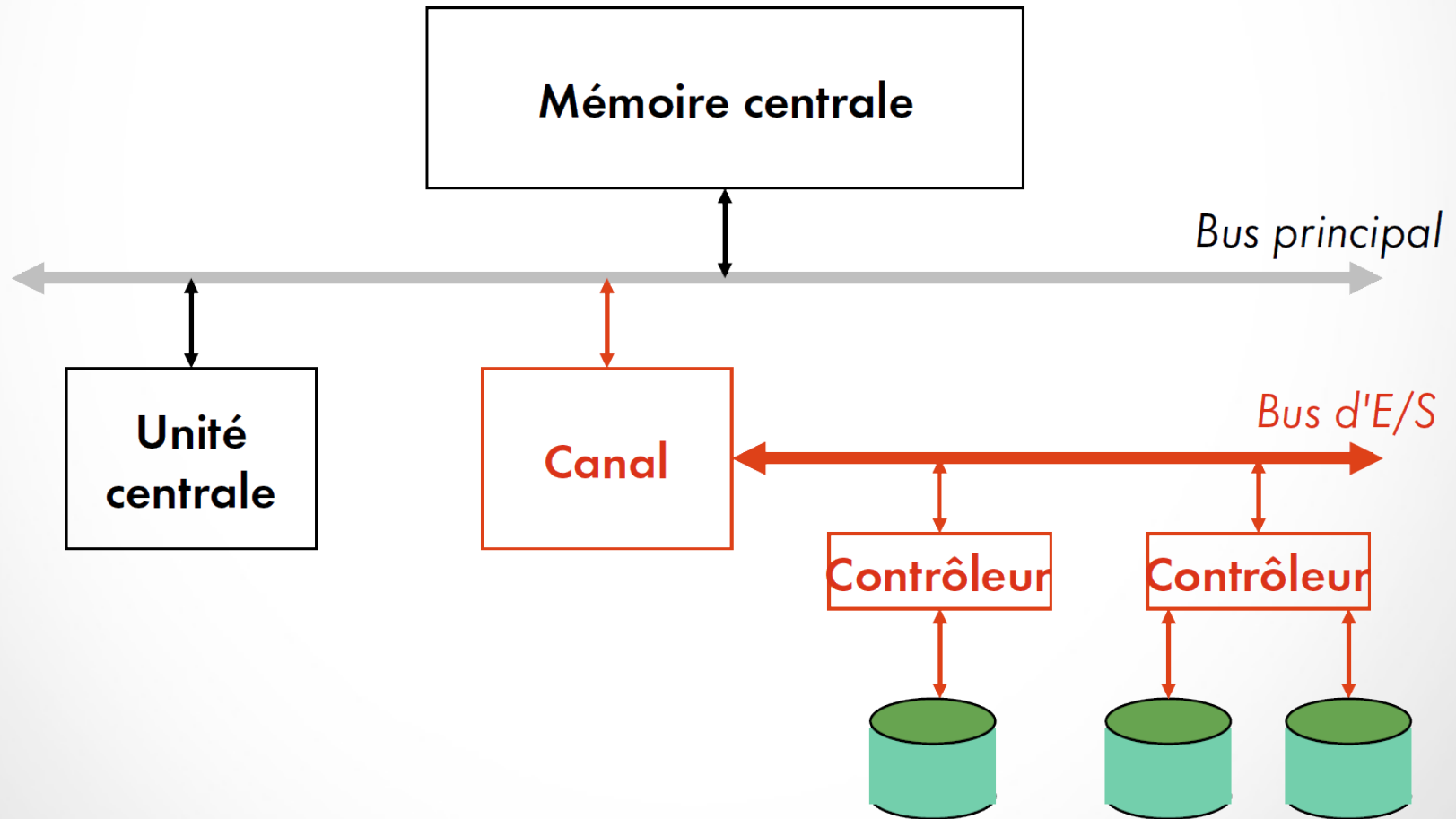
- Processeur spécialisé d'E/S

✓ Pilote (« driver »)

- Programme d'interface entre le système d'exploitation et le système d'E/S



Structure du Système d'E/S

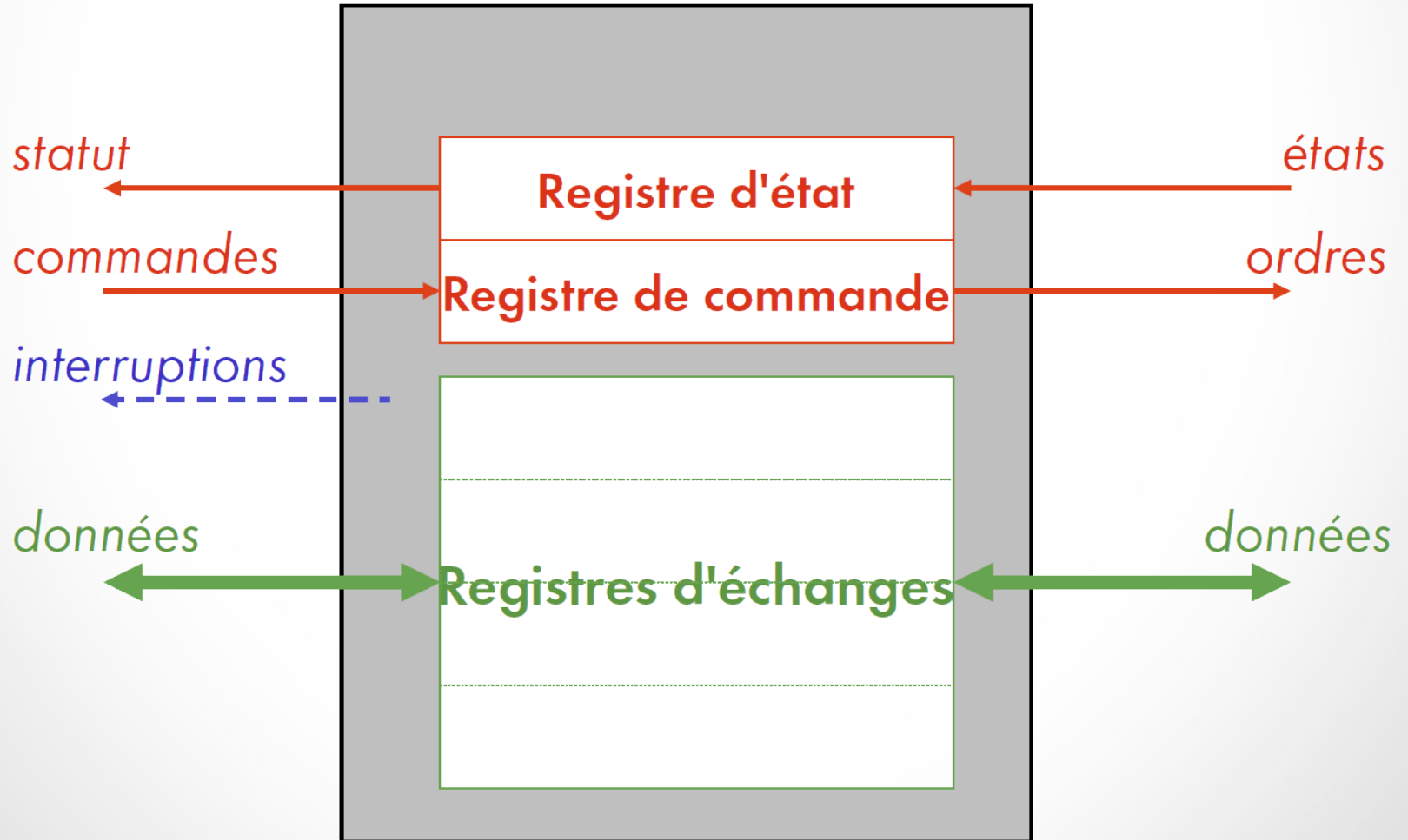


Structure du Système d'E/S : Contrôleur

Unité centrale

Contrôleur

Périphérique





Adressage des Périphériques

✓ Adressage et instructions spécifiques

- Espace d'adressage propre
 - canal, contrôleur, unité
- Instructions spécifiques d'E/S
- `start_io`
- `sense_status`
- `read, write`

✓ E/S « mappées »

- Les registres des contrôleurs ont des adresses dans l'espace mémoire virtuel du noyau du système d'exploitation
- Les programmes (« drivers ») y accèdent donc par de simples lecture/écriture en mémoire
 - Mais la sémantique en est très différente

Phases successives d'une Entrée-Sortie

✓ Initialisation

- Paramétrage du contrôleur, périphérique...
- Attente de fin d'initialisation

✓ Entretien de l'échange

- Échange des données entre le contrôleur et l'unité centrale ou la mémoire
- Rythme donné par le périphérique

✓ Nettoyage final

Modes de Gestion : Modes Programmés

✓ Mode programmé simple

- L'UC est entièrement responsable de l'échange
- Le transfert entre mémoire et périphérique passe à travers les registres de l'UC
- Attente active entre les phases

✓ Mode programmé par interruptions

- Le contrôleur interrompt l'UC quand il est prêt
 - pour le début de l'échange
 - à chaque unité d'échange
- Le sous-programme de traitement de l'IT assure l'entretien de l'échange
- Le transfert entre mémoire et périphérique passe à travers les registres de l'UC
- UC libre entre deux unités d'échange

Mode Programmé Simple: Attente Active (polling)

✓ Mode programmé simple

– Exemple de lecture

```
initialisation  
préparation du premier échange  
tant que quelque chose à lire  
faire  
    attendre périphérique prêt  
    vérifier registre d'état  
    lire registre(s) échange(s)  
    préparer échange suivant  
fin faire  
nettoyage final
```

✓ Simple

✓ Lent

✓ Monopolise l'UC

✓ Utilisé pour le
chargement initial du
système (« boot »)

Mode Programmé par Interruptions

✓ Mode programmé par IT

– Exemple de lecture

Driver (mode processus)

*initialisation
préparation du 1^{er} échange*

Unité centrale
disponible
(sauf pendant
les traitements d'IT)

nettoyage final

Driver (mode interruption)

*vérifier registre d'état
lire registre(s) échange(s)
si quelque chose à lire
alors
 préparer échange suivant
sinon
 réveiller le processus
fin si
retour d'interruption*

IT

✓ Assez simple

✓ L'UC est libre entre deux échanges

✓ Risque de perte d'interruption

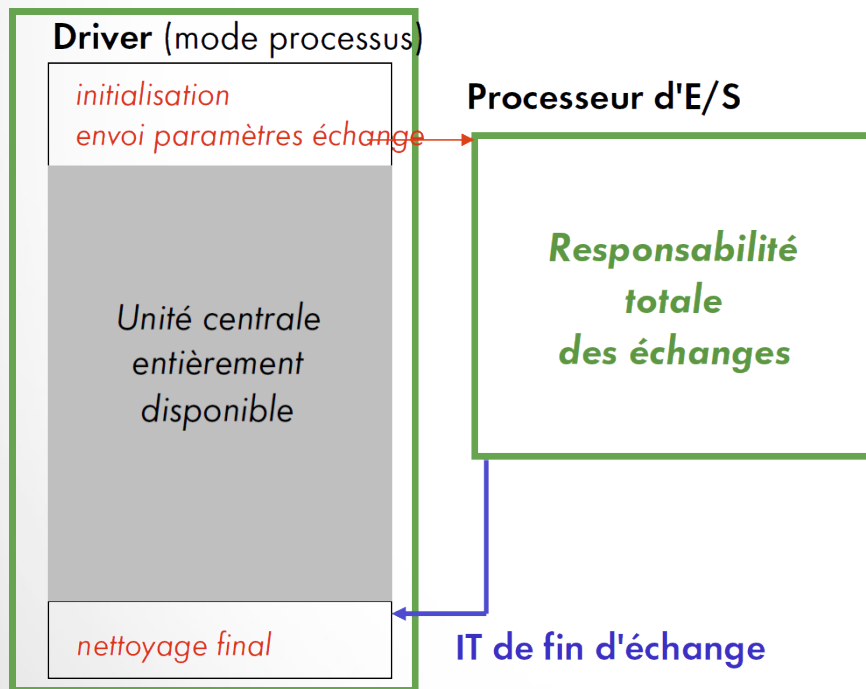
✓ Convient aux périphériques lents

– Clavier, souris, ...



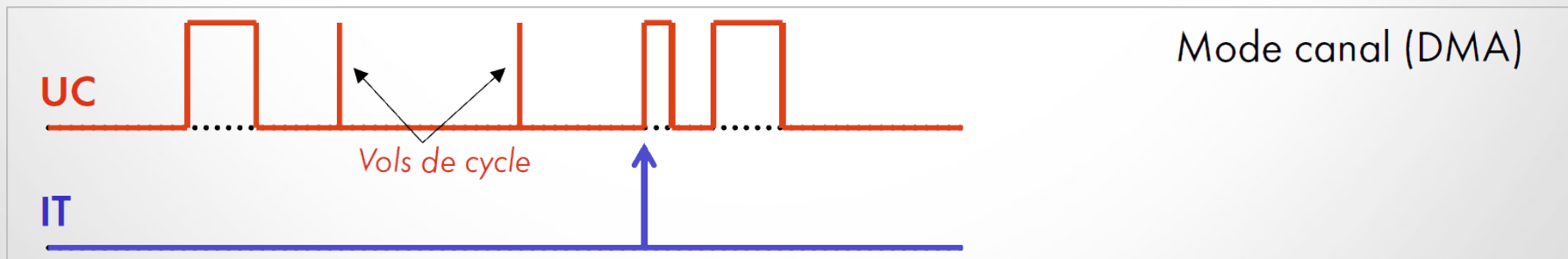
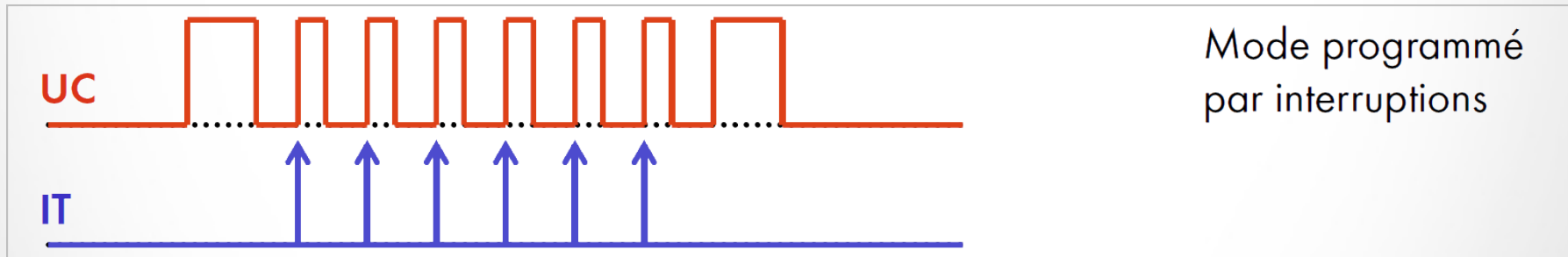
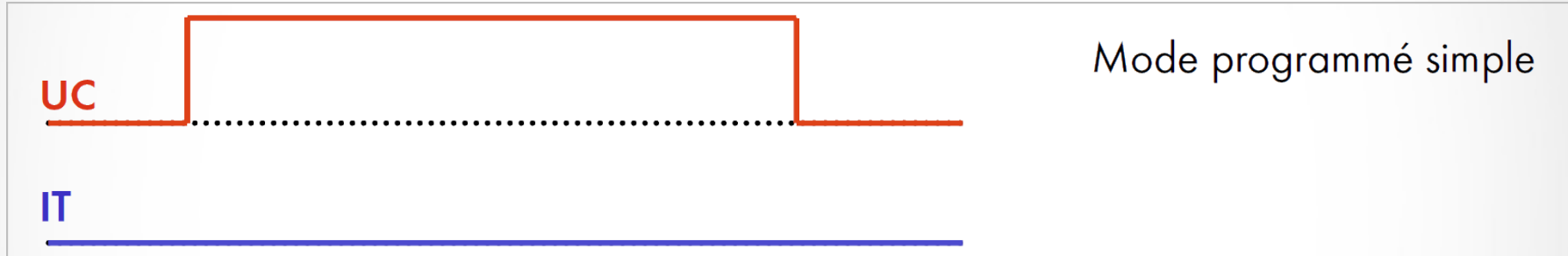
Modes de gestion des E/S : Mode Canal (DMA)

- ✓ L'UC est totalement libre entre deux échanges
- ✓ Possibilité de « cache » dans le processeur d'E/S
- ✓ Convient aux périphériques rapides
 - Disques
 - Acquisition de données
 - Réseaux rapides





Modes de gestion des E/S : Utilisation de l'UC





Gestion des Entrées-Sorties

Suivant la norme Posix.1

Fichiers et Répertoires: Modèle

- ✓ **Modèle hiérarchique**
 - Répertoires / Dossiers

- ✓ **Types de fichiers**
 - ordinaire
 - données textuelles, binaires...
 - répertoire
 - spéciaux (périphériques)
 - modes bloc et caractères (voir dans /dev)
 - fichier FIFO (« pipe nommé »)
 - communication (sockets...)
 - etc.



Répertoire Courant

✓ `#include <unistd.h>`

✓ **Changement de répertoire**

```
int chdir(const char *path);
```

✓ **Consultation du répertoire courant**

```
char *getcwd(char *dirname, size_t size);
```

- **size est la taille du tableau pointé par `dirname` (et qui a dû être alloué par l'appelant)**

Consultation des Attributs d'un Fichier 1/3

✓ La fonction `stat`

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *path, struct stat *buf);
```

✓ Structure `stat`

- `st_dev`, `st_ino` : **périphérique et numéro du fichier**
- `st_mode` : **type et droits d'accès**
- `st_nlink` : **nombre de liens**
- `st_uid`, `st_gid` : **propriétaire et son groupe**
- `st_size` : **taille du fichier (nombre de caractères)**
- `st_atime` : **date du dernier accès,**
- `st_mtime` : **date de la dernière modification,**
- `st_ctime` : **date du dernier changement d'attributs**

Consultation des Attributs d'un Fichier 2/3

- ✓ **Macros de décodage de `st_mode`**
- ✓ **Sélection des champs**
 - `S_IRWX[UGO]` `S_IS[UG]ID`
- ✓ **Décodage des droits d'accès**
 - `S_I[RWX] (USR | GRP | OTH)`
- ✓ **Type du fichier (argument `m : st_mode`)**
 - `S_ISCHR(m)` : Teste si `m` est un pilote de type caractère (char)
 - `S_ISBLK(m)` : Teste si `m` est un pilote de type bloc (block)
 - `S_ISDIR(m)` : Teste si `m` est un répertoire/dossier (directory)
 - `S_ISREG(m)` : Teste si `m` est un fichier normal (regular)
 - `S_ISFIFO(m)` : Teste si `m` est une fichier « d'échange FIFO » (fifo)
 - **Cf: Communication inter-processus que nous verrons plus tard**

Consultation des Attributs d'un Fichier 3/3

✓ Exemple

```
struct stat sbuf;
char *path = "foo/bar";
if (stat(path, &sbuf) >= 0) {
    int m = sbuf.st_mode;
    if (S_ISREG(m)) {
        /* le fichier est un fichier ordinaire */ ...
        if (m & (S_IWUSR | S_IWGRP)) {
            /* le fichier est inscriptible par le
               propriétaire ou son groupe */ ...
        }
    }
}
```

Lecture des Répertoires

1/2

✓ Fonctions de lecture

```
#include <sys/types.h>
#include <dirent.h>
```

```
DIR *opendir(const char *dirpath);
struct dirent *readdir(DIR *dirp);
int closedir(DIR *dirp);
int rewinddir(DIR *dirp);
```

✓ Champs de struct dirent

- **un seul champ portable, `d_name`, tableau de caractères de dimension `NAME_MAX`**

Lecture des Répertoires

2/2

✓ Exemple : une version rustique de `ls`

```
struct dirent *dentry;
DIR *dirp = opendir("foo");
if (dirp == NULL) {
    perror("Répertoire inaccessible\n");
    exit(EXIT_FAILURE);
}
while ((dentry = readdir(dirp)) != NULL) {
    printf("%s\n", dentry->d_name);
}
closedir(dirp);
```

Manipulation de Répertoires

1/2

✓ Création/destruction de répertoires

```
#include <sys/types.h>
#include <sys/stat.h>
int mkdir(const char *path, mode_t mode);
int rmdir(const char *path)
```

- mode **ne doit contenir que des droits d'accès**
- **on ne peut détruire qu'un répertoire vide**

✓ Exemple

```
mkdir("foo", S_IRUSR | S_IWUSR | S_IRGRP);
```

✓ Création et destruction d'entrées dans un répertoire

```
#include <unistd.h>
int link(const char *oldpath, const char *newpath);
int unlink(const char *path);
int rename(const char *oldpath, const char *newpath);
```



Bibliothèque ANSI C

- ✓ **Posix.1** contient l'ensemble de la bibliothèque d'entrée-sortie standard de C
 - `<stdio.h>`
- ✓ **Les E/S ont lieu avec « bufferisation »**
 - sauf si le support est un terminal
 - la fonction `fflush(FILE *)` vide le buffer
- ✓ **En fait ces fonctions d'E/S sont réalisées avec des fonctions primitives qui constituent l'API d'E/S de base de Posix**



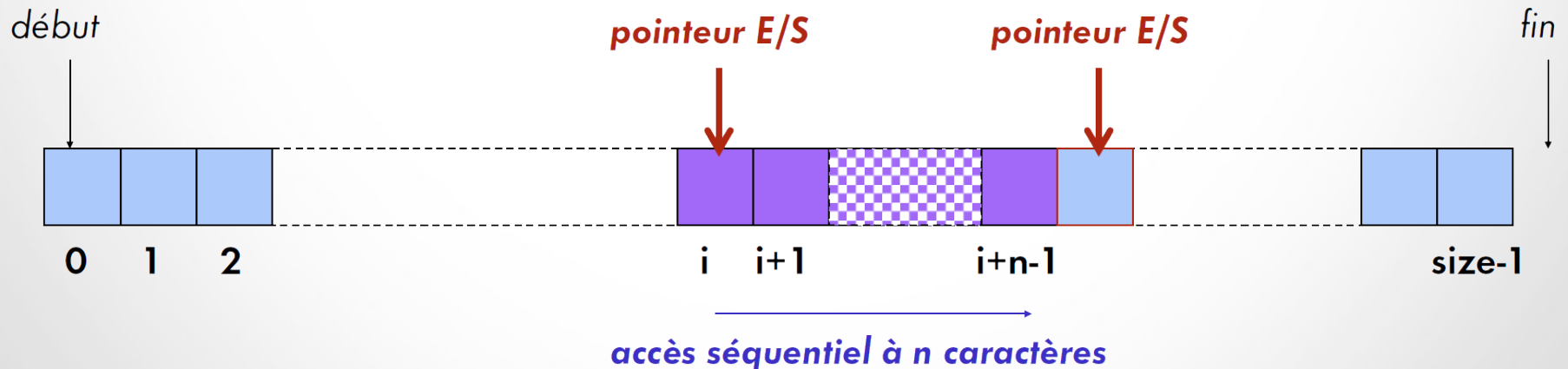
Descripteur de fichiers

- ✓ Un « **file descriptor** », `fd`, est un numéro d'unité logique qui permet de référencer un fichier
- ✓ `fd` est un entier positif ou nul
 - 0, 1 et 2 correspondent respectivement à l'entrée, la sortie et l'erreur standards
- ✓ Les « **file descriptors** » sont alloués par processus
- ✓ Les fonctions qui retournent un descripteur de fichier (e.g., `open`) retournent en général la plus petite valeur disponible dans le processus courant



Modèle de base des Fichiers

- ✓ **Tableau de caractères**
 - indexé à partir de 0
- ✓ **Pointeur d'E/S**
 - index courant
 - manipulable directement (accès direct)
- ✓ **Lecture/écriture séquentielle**
 - à partir de l'index courant



Ouverture et Création de Fichiers 1/2

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
int open(const char *path, int oflag[, mode_t mode]);
```

✓ **Flag** oflag

- O_RDONLY, O_WRONLY, O_RDWR : **mode d'ouverture**
- O_APPEND : **place le pointeur d'E/S en fin de fichier**
- O_CREAT : **crée le fichier s'il n'existe pas**
- O_EXCL **si** O_CREAT **et si le fichier existe, erreur**
- O_TRUNC : **tronquer le fichier s'il existe**
- **etc.**

Ouverture et Création de Fichiers 2/2

- ✓ **Troisième argument de `open` (mode)**
 - utilisé seulement si `O_CREAT`
 - positionne les droits d'accès au nouveau fichier
 - filtrage par le `UMASK`

- ✓ **Exemple**

```
if (open( "foo", O_WRONLY | O_CREAT | O_TRUNC,  
        S_IRUSR | S_IWUSR | S_IRGRP ) < 0 ) {  
    perror( "open" );  
}
```

Masque de Création de Fichiers (cmask)

```
#include <sys/types.h>
#include <sys/stat.h>
```

```
mode_t umask(mode_t mode);
```

- **mode ne doit contenir que des droits d'accès**
- **la fonction retourne la valeur précédente du masque**
- **mode contient les droits que l'on veut dénier**
 - **après** `fd = open("foo", ... | O_CREAT, m);`
 - **le mode est** : `m & ~cmask`
- **le masque est un attribut du processus**

✓ Exemple

```
oldmask = umask(S_IWGRP | S_IWOTH);
```

Lecture/Ecriture Séquentielles

1/2

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

```
ssize_t read(int fd, void *buf, size_t n);  
ssize_t write(int fd, void *buf, size_t n);
```

✓ E/S séquentielles

- Le pointeur d'E/S est avancé du nombre de caractères effectivement transférés
- Par défaut, `read()` est bloquant ; `write()` peut l'être

✓ Valeur de retour

- Nombre de caractères effectivement transférés
- Pour `read()`, valeur de retour 0 \Rightarrow fin de fichier

Lecture/Ecriture Séquentielles

2/2

- ✓ **Exemple : une version rustique et partielle de copie de fichiers**

```
char buffer[MAX];  
int n;  
int fdin = open("foo", O_RDONLY | O_EXCL);  
int fdout =  
    open("bar", O_WRONLY | O_TRUNC | O_CREAT, ...);  
  
while((n = read(fdin, buffer, MAX)) != 0)  
    write(fdout, buffer, n);
```

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

```
off_t lseek(int fd, off_t offset, int whence);
```

✓ Manipulation directe du pointeur d'E/S

- **Nouvelle position : $\text{Orig}(\text{whence}) + \text{offset}$**
 - **$\text{Orig}(\text{SEEK_SET})$: début du fichier (0)**
 - **$\text{Orig}(\text{SEEK_CUR})$: position courante**
 - **$\text{Orig}(\text{SEEK_EOF})$: fin de fichier**
- **Valeur de retour : nouvelle position absolue (i.e., depuis début du fichier)**

✓ Exemples:

```
off_t off = lseek(fd, 0, SEEK_CUR);
```

- **Retourne la position absolue courante, sans la modifier ($\text{lseek}(\text{fd})$)**

```
lseek(fd, 0, SEEK_SET);
```

- **Retourne au début du fichier ($\text{rewind}(\text{fd})$)**

```
off_t off = lseek(fd, 100, SEEK_EOF);
```

- **Ajoute 100 caractères (nuls) après la fin de fichier**
- **Le fichier doit avoir été ouvert en écriture**

✓ Exemple : accès direct à une structure de données

```
struct Data { ... } buf;  
int i = ... ;
```

```
/* accès au ième élément et lecture */  
lseek(fd, i*sizeof(struct Data), SEEK_SET);  
read(fd, &buf, sizeof(struct Data));
```

Mélange avec la Bibliothèque ANSI C

✓ Descripteur de fichier → FILE *

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

```
int fd;
```

```
FILE *fp = fdopen(fd, type);
```

– **voir** `fopen()` **pour la signification de type**

✓ FILE * → **descripteur de fichier**

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

```
FILE *fp;
```

```
int fd = fileno(fp);
```