Entrées/sorties

Généralités

Entrées/sorties

Les entrées/sorties permettent de communiquer avec l'environnement.
Elles peuvent se faire :

Entrées/sorties

Les entrées/sorties permettent de communiquer avec l'environnement.

Elles peuvent se faire :

- soit par les flux standard
  - entrée standard (*stdin, 0, System.in, std::cin, . . .*)
  - sortie standard (*stdout, 1, System.out, std::cout . . .*)
  - sortie d'erreur standard (*stderr, 2, System.err, std::cerr, . . .*)

Entrées/sorties

Les entrées/sorties permettent de communiquer avec l'environnement.
Elles peuvent se faire :

- soit par les flux standard
  - entrée standard (*stdin, 0, System.in, std::cin, . . .*)
  - sortie standard (*stdout, 1, System.out, std::cout . . .*)
  - sortie d'erreur standard (*stderr, 2, System.err, std::cerr, . . .*)
- soit avec un descripteur de fichier associé à un fichier *ouvert* pour des opérations de *lecture* ou d'*écriture*.

Interfaces de programmation

Deux interfaces de programmation (API) peuvent être utilisées.

Interfaces de programmation

Deux interfaces de programmation (API) peuvent être utilisées.

Haut-niveau

- fournie par la bibliothèque standard (*libc*)
- indépendante du système d'exploitation
- descripteur de fichier de type `FILE*`

Interfaces de programmation

Deux interfaces de programmation (API) peuvent être utilisées.

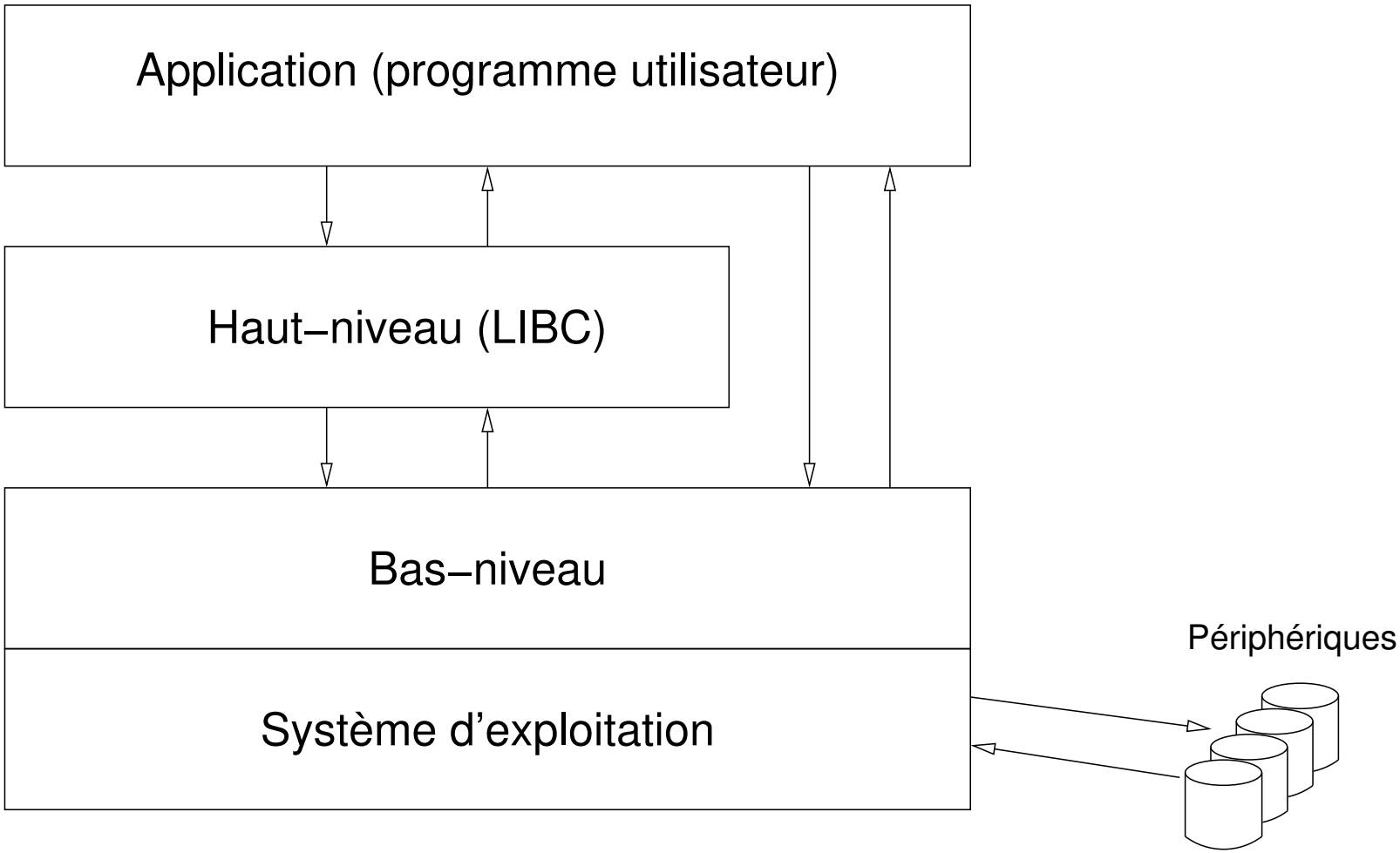
Haut-niveau

- fournie par la bibliothèque standard (*libc*)
- indépendante du système d'exploitation
- descripteur de fichier de type `FILE*`

Bas-niveau

- fournie par le système d'exploitation
- offre plus de fonctionnalités (ex : permissions, tubes, etc.)
- descripteur de fichier de type entier (`int`)

Organisation en couches



Haut-niveau

E/S de haut-niveau

- Réalisées par de nombreuses fonctions de la bibliothèque C standard.
- Déclarées dans *stdio.h* :

  #include <stdio.h>
- On peut séparer les fonctions en deux classes :
  - celles réalisant les opérations sur les flux standards (ex : printf, scanf, getchar, putchar, . . .)
  - celles nécessitant de passer en paramètre le descripteur du fichier (ex : fprintf, fscanf, fread, fwrite, . . .)
- Descripteur de fichier de type : `FILE*`
- Flux standards : `stdin`, `stdout`, `stderr`
- Documentation : manuel en ligne (ex : `man 3 fopen`)

Ouverture de fichier

L'opération d'*ouverture de fichier* permet d'obtenir un descripteur associé à un fichier pour y faire des opérations.

```
FILE *fopen(const char *pathname, const char *mode);
FILE *fdopen(int fd, const char *mode);
FILE *freopen(const char *pathname, const char *mode, FILE *stream);
```

- fopen ouvre un fichier
- fdopen ouvre à haut niveau le descripteur de bas niveau valide *fd*
- freopen ouvre le fichier et l'associe au descripteur *stream* (redirections)

Chacune des fonctions retourne un pointeur non NULL en cas de succès, NULL en cas d'échec.

Paramètres de fopen

pathname	nom du fichier à ouvrir
mode	mode d'ouverture
"r"	lecture
"r+"	lecture et écriture
"w"	écriture ; le fichier est créé ou tronqué s'il existe
"w+"	lecture et écriture ; le fichier est créé ou tronqué s'il existe
"a"	écriture en ajout ; le fichier est créé
"a+"	lecture au début, écriture en ajout ; le fichier est créé

Exemples

```
FILE* f;
/* Ouverture de toto.txt, du répertoire courant, en lecture */
f = fopen("toto.txt", "r");
if (f == NULL) {
    fprintf(stderr, "Echec de l'ouverture du fichier toto.txt\n");
    return -1;
}
```

Exemples

```
FILE* f;
/* Ouverture de toto.txt, du répertoire courant, en lecture */
f = fopen("toto.txt", "r");
if (f == NULL) {
    fprintf(stderr, "Echec de l'ouverture du fichier toto.txt\n");
    return -1;
}
```

```
/* Ouverture de toto.txt, pour rediriger stdout */
f = freopen("toto.txt", "w", stdout);
if (f == NULL) {
  fprintf(stderr, "Echec de l'ouverture du fichier toto.txt\n");
  return -1;
}
```

Fermeture du fichier

```
int fclose(FILE *stream);
```

Ferme le descripteur stream et retourne 0 en cas de succès EOF en cas d'échec.

Lectures/écritures non formatées

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

- Lecture (fread) de *nmemb* objets de taille *size* depuis le fichier de descripteur *stream*. Les données sont enregistrées à l'adresse *ptr*.
- Écriture (fwrite) de *nmemb* objets de taille *size* vers le fichier de descripteur *stream*. Les données sont obtenues à l'adresse *ptr*.
- Résultat retourné : le nombre d'objets lus ou écrits, 0 en cas d'erreur ou fin de fichier.

Exemples

```
/* Lecture de 50 caractères ou octets */
char ligne[50];
n = fread(ligne, 1, 50, f); // NB: sizeof(char) == 1
if (n != 50) {
    /* message */
}
```

Exemples

```
/* Lecture de 50 caractères ou octets */
char ligne[50];
n = fread(ligne, 1, 50, f); // NB: sizeof(char) == 1
if (n != 50) {
    /* message */
}
```

```
/* Lecture de 50 entiers */
int t[50];
n = fread(t, sizeof(int), 50, f);
if (n != 50) {
    /* message */
}
```

Autres fonctions

```
void clearerr(FILE *stream);
int feof(FILE *stream);
int ferror(FILE *stream);
```

- La fonction clearerr efface les indicateurs de fin de fichier et d'erreur.
- La fonction feof teste si l'indicateur de fin de fichier est positionné.
- La fonction ferror teste si l'indicateur d'erreur est positionné.

Lectures/écritures par caractères :

- *fgetc, getc, getchar, fgets, ungetc*
- *fputc, putc, putchar, fputs, puts*

Écritures formatées

```
int printf(const char *format, ...);
int fprintf(FILE *stream, const char *format, ...);
int sprintf(char *str, const char *format, ...); // fortement déconseillé
int snprintf(char *str, size_t size, const char *format, ...);
```

- *printf* écrit sur la sortie standard (stdout)
- *fprintf* écrit sur le descripteur *stream*
- *sprintf* et *snprintf* écrivent une chaîne en mémoire (pointeur *str*)

Lectures formatées

```
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
int sscanf(const char *str, const char *format, ...);
```

- *scanf* lit depuis l'entrée standard (stdin)
- *fscanf* lit depuis le descripteur *stream*
- *sscanf* lit depuis une chaîne en mémoire (pointeur *str)*

Exemples

```
int i;
float t[128];
printf("Donnez la valeur entière de i");
scanf("%d", &i);
printf("Donnez la valeur réelle de t[0]");
scanf("%f", t); // ou scanf("%f", &t[0]);
```

Exemple : copie par caractère

```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    const char *fichier_source = "toto.txt";
    FILE *fich = fopen(fichier_source, "r");
    if (fich == NULL) {
        perror("fopen");
        return EXIT_FAILURE;
    }
    int c = getc(fich);
    while (c != EOF) {
        putchar(c);
        c = getc(fich);
    }
    int err = fclose(fich);
    return err ? EXIT_FAILURE : EXIT_SUCCESS;
}
```

Bas niveau

E/S de bas-niveau

- Réalisées par des primitives du système d'exploitation.
- Nombreux fichiers d'en-tête : se référer au manuel
- Descripteur de fichier : entier positif (type `int`)
  - attribués par ordre croissant (premier disponible)
- Flux standards : `STDIN_FILENO` ($= 0$), `STDOUT_FILENO` ($= 1$), `STDERR_FILENO` ($= 2$)
- Documentation : manuel en ligne (ex : `man 2 open`)

Ouverture de fichier

```
#include <fcntl.h>
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int creat(const char *pathname, mode_t mode);
```

- ouvrent ou créent le fichier désigné
- retourne le descripteur d'accès au fichier en cas de succès, -1 en cas d'erreur.

Paramètres de open

pathname nom du fichier à ouvrir

flags mode d'ouverture, composé (*ou binaire*) à partir des constantes

- `O_RDONLY` lecture seule
- `O_WRONLY` écriture seule
- `O_RDWR` lecture et écriture

Paramètres de open

pathname nom du fichier à ouvrir

flags mode d'ouverture, composé (*ou binaire*) à partir des constantes

- `O_RDONLY` lecture seule
- `O_WRONLY` écriture seule
- `O_RDWR` lecture et écriture

auxquelles on peut ajouter

- `O_APPEND` écriture en mode ajout
- `O_CREAT` le fichier est créé s'il n'existe pas
- `O_EXCL` accès exclusif (utilisé avec `O_CREAT`) : échec si le ficher existe déjà
- `O_TRUNC` le contenu existant est détruit

Paramètres de open (suite)

mode droits à donner au ficher créé, combinaison de constantes

- `S_IRWXU` (00700) read, write, execute by owner
- `S_IRUSR` (00400) read by owner
- `S_IWUSR` (00200) write by owner
- `S_IXUSR` (00100) execute/search by owner
- `S_IRWXG` (00070) read, write, execute by group
- `S_IRGRP` (00040) read by group
- `S_IWGRP` (00020) write by group
- `S_IXGRP` (00010) execute/search by group
- `S_IRWXO` (00007) read, write, execute by others
- `S_IROTH` (00004) read by others
- `S_IWOTH` (00002) write by others
- `S_IXOTH` (00001) execute/search by others

Variantes de open

- la variante avec trois arguments (*mode*) doit être utilisée si la création de fichier est autorisée (`O_CREAT`)
- un appel à `creat()` est équivalent à appeler `open()` avec *flags* égal à `O_WRONLY|O_CREAT|O_TRUNC`

Fermeture de fichier

```
#include <unistd.h>
int close(int fd);
```

- ferme le descripteur *fd*
- retourne 0 en cas de succès, -1 en cas d'erreur

Lectures/écritures

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
```

- Lecture (read) de *count* octets depuis le fichier de descripteur *fd*. Les données sont enregistrées à l'adresse *buf*.
- Écriture (write) de *count* octets vers le fichier de descripteur *fd*. Les données sont obtenues à l'adresse *buf*.
- Résultat retourné : le nombre d'octets lus ou écrits, -1 en cas d'erreur.

NB : d'autres fonctions sont disponibles (ex : lseek pour changer la position dans le fichier)

Exemple (1/3)

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

#define NB 512 // taille d'un bloc de lecture

int main(void)
{
    char *erreur_ouverture = "Erreur d'ouverture du fichier fichier.txt\n";
    char tampon[NB];
    int fich, nlus, necrit;

    fich = open("/tmp/fichier.txt", O_RDONLY);
    if (fich == -1) {
        write(STDERR_FILENO, erreur_ouverture, strlen(erreur_ouverture));
        return -1;
    }
    /* ... */
```

Exemple (2/3)

```
/* ... */
/* Lecture NB premiers caractères du fichier dans tampon */
nlus = read(fich, tampon, NB);
while (nlus != 0) { /* tant que non fin de fichier */
    if (nlus == -1) {
        /* Une erreur s'est produite durant la lecture */
        /* écriture du message système correspondant à l'erreur */
        perror("Erreur de lecture de fichier.txt");
        close (fich);
        return -1;
    } /* nlus == -1 */
    /*
       Il n'y a pas eu d'erreur de lecture
       Le nombre de caractères lus est > 0 et <= NB
       Écriture des caractères lus sur la sortie standard
     */
    necrit = write (1, tampon, nlus);
    /* ... */
```

Exemple (3/3)

```
        /* ... */
        if (necrit != nlus) {
            fprintf(stderr, "Le nombre de caractères écrits : %d est "
                    "différent du nombre de caractères lus %d\n", necrit, nlus);
            /*
               Ce n'est pas une erreur, il faudrait écrire les caractères restants
             */
            close (fich);
            return 0;
        }
        nlus = read(fich, tampon, NB);
    }
    /* Fermeture du fichier */
    close (fich);
    return 0;
}
```

Synthèse

Synthèse

	Haut niveau	**Bas niveau**
Descripteur de fichier	FILE*	int
Entrée standard	stdin	0 ou STDIN_FILENO
Sortie standard	stdout	1 ou STDOUT_FILENO
Sortie d'erreur standard	stderr	2 ou STDERR_FILENO
Ouvrir un fichier	fopen	open
Fermer un fichier	fclose	close
Lecture non formatée par bloc	fread	read
Écriture non formatée par bloc	fwrite	write
Lecture formatée	fscanf	*n/a*
Écriture formatée	fprintf	*n/a*
Manuel en ligne	man 3 ...	man 2 ...