Langage C, Le monde extérieur

Système et environnement de programmation

Université Grenoble Alpes

Préliminaire : options utiles de `clang`

- `-o <nom>` permet de choisir le nom de l'exécutable (attention, écrase la version précédente)
- `-Wall` demande à `clang` d'afficher plus d'avertissements
- `-Werror` considère les avertissements comme des erreurs

Routine de compilation/exécution typique :

```
clang -Wall monprog.c -o monprog
./monprog
```

Plan

Passer des arguments sur la ligne de commande

Beaucoup de commandes acceptent des arguments, par exemple

```
ls -l
mv source destination
cp -r source destination
```

De manière analogue, nos programmes en C peuvent en recevoir

```
./a.out mes arguments sont au nombre de 7
```

Dans le programme

Ils sont transmis au main sous la forme suivante

- un entier : le nombre d'arguments (commande comprise)
- un tableau de chaînes de caractères : les arguments

```
int main(int argc, char *argv[]) {
    printf("Commande : %s\n", argv[0]);
    printf("%d argument(s) reçus :\n", argc-1);
    for (int i=1; i<argc; i++) {
        printf("- argument %d : %s\n",
               i, argv[i]);
    }
    return 0;
}
```

Exécution

```
clang exemple_arguments.c -o exmp_args
./exmp_args un exemple avec des arguments, et pas 0
```

```
Commande : ./exmp_args
8 argument(s) reçus :
- argument 1 : un
- argument 2 : exemple
- argument 3 : avec
- argument 4 : des
- argument 5 : arguments,
- argument 6 : et
- argument 7 : pas
- argument 8 : 0
```

Type

Attention, les arguments de la ligne de commande sont des chaînes de caractères

- conversion de la représentation d'une valeur d'un autre type
- exemple de représentations ayant la même valeur entière
  - 42
  - 0x2a
  - 052
  - 0b101010
- on peut utiliser `sscanf` qui lit depuis une chaine

Exemple

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

    for (int i=1; i<argc; i++) {
        printf("L'argument %d (%s) ", i, argv[i]);
        int resultat;
        resultat = sscanf(argv[i], "%d", &valeur);
        if (resultat == 1) {
            printf("vaut %d\n", valeur);
        } else {
            printf("n'est pas entier\n");
        }
    }
    return 0;
}
```

Exécution

```
clang exemple_argument_entier.c -o ex_arg_ent
./ex_arg_ent exemple_1, partie 4a avec 17, 42 et 0
```

```
L'argument 1 (exemple_1,) n'est pas entier
L'argument 2 (partie) n'est pas entier
L'argument 3 (4a) vaut 4
L'argument 4 (avec) n'est pas entier
L'argument 5 (17,) vaut 17
L'argument 6 (42) vaut 42
L'argument 7 (et) n'est pas entier
L'argument 8 (0) vaut 0
```

Plan

Définition

Un fichier est un élément de stockage contenant de l'information

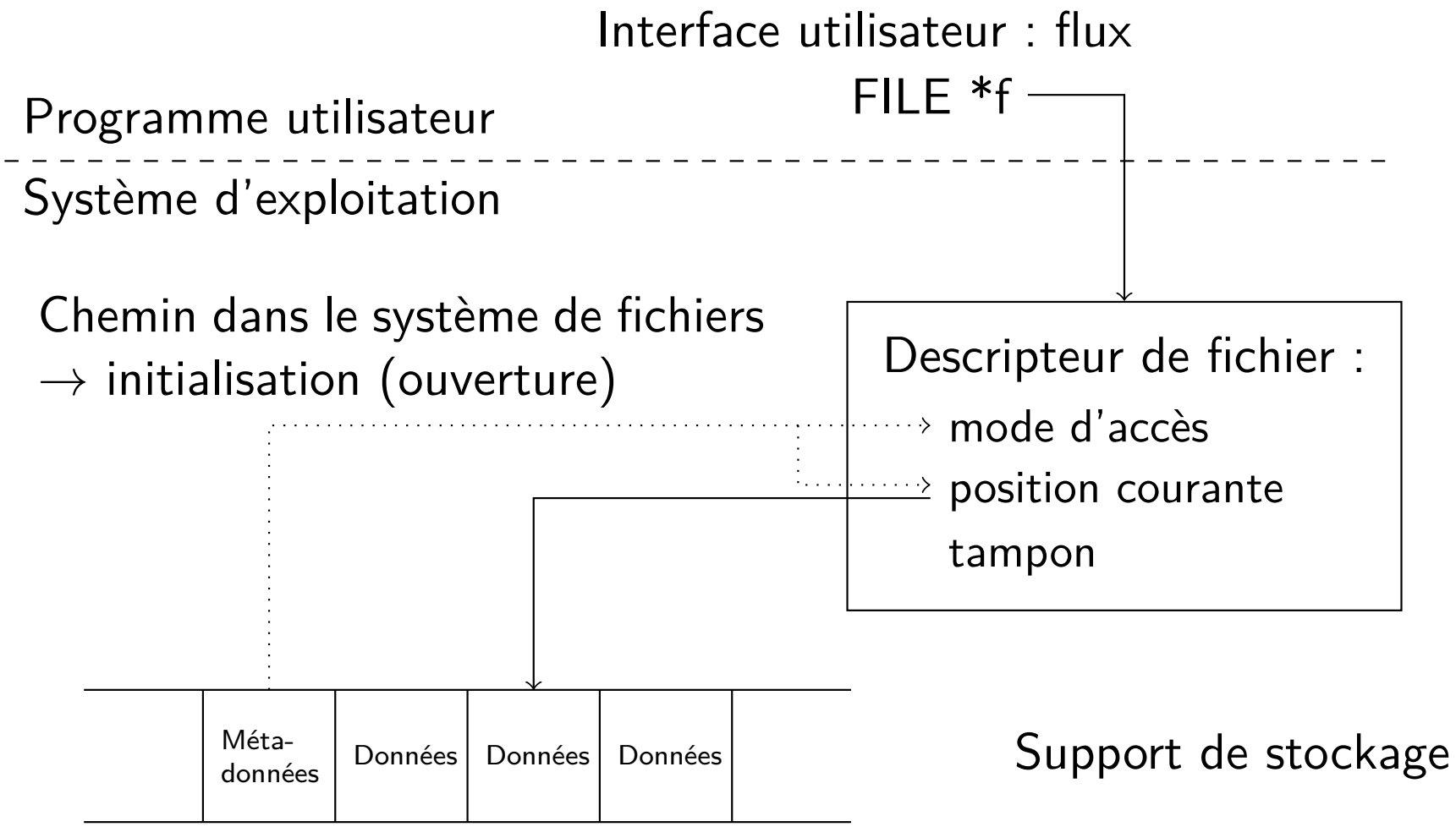
- abstraction fournie par le système
- a la forme d'une séquence d'octets

Le programmeur y accède selon un seul des deux modes
(nous éviterons les modes mixtes qui compliquent les choses)

- lecture
- écriture

Chaque accès avance dans la séquence

Principe général



Interface utilisateur (stdio.h)

Séquence d'actions à respecter

1. ouverture du fichier : initialise le descripteur

```
FILE *fopen(char *nom, char *mode);
```

2. accès au fichier : lectures ou écritures selon le mode (ATTENTION : chaque accès fait avancer la position)

```
int fprintf(FILE *flux, char *format, ...);
int fscanf(FILE *flux, char *format, ...);
```

3. test de fin : après l'echec d'un accès en lecture (invalide donc)

```
int feof(FILE *flux);
```

4. fermeture du fichier : libère le descripteur

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

Descripteurs particuliers

Ils sont déjà initialisés par le système pour tout programme

- `stdin` : accessible en lecture, correspond au clavier
- `stdout` : accessible en écriture, correspond à l'écran
- `stderr` : accessible en écriture, correspond à l'écran

Permettent de factoriser le code

- `scanf(...)` $\Longleftrightarrow$ `fscanf(stdin, ...)`
- `printf(...)` $\Longleftrightarrow$ `fprintf(stdout, ...)`

$\Rightarrow$ on écrit une fois le code, il s'applique à la fois aux fichiers et à l'écran/clavier

Exemple : affichage du contenu d'un fichier

```c
int main() {
    char nom[128] = "toto.txt";
    FILE *f; char c;

    f = fopen(nom, "r");
    if (f == NULL) {
        perror(nom); // affichage erreur système
        exit(1);
    }
    fscanf(f, "%c", &c);
    while (!feof(f)) {
        printf("%c", c);
        fscanf(f, "%c", &c);
    }
    fclose(f);
    return 0;
}
```

Avec le nom passé en argument de la ligne de commande

```
int main(int argc, char *argv[]) {
    FILE *f; char c;

    if (argc < 2) {
        fprintf(stderr, "Erreur, "
                "pas assez d'arguments\n");
        exit(2);
    }
    f = fopen(argv[1], "r");
    if (f == NULL) {
        perror(argv[1]); // affichage erreur système
        exit(1);
    }

// ...
```

En choisissant clavier ou fichier de nom donné

```
int main(int argc, char *argv[]) {
    FILE *f; char c;

    if (argc < 2) {
        f = stdin;
    } else {
        f = fopen(argv[1], "r");
        if (f == NULL) {
            perror(nom); // affichage erreur système
            exit(1);
        }
    }

// ...
```

Exemple : écriture dans un fichier

```c
int main() {
    char nom[128] = "toto.txt";
    FILE *f;

    f = fopen(nom, "w");
    if (f == NULL) {
        perror(nom); // affichage erreur système
        exit(1);
    }

    for(int i=1; i<=100; i++) {
        fprintf(f, "%d\n", i);
    }
    fclose(f);
    return 0;
}
```

Avec le nombre d'entiers à écrire passé en argument

```
int main(int argc, char *argv[]) {
    char nom[128] = "toto.txt";
    FILE *f; int N;

    f = fopen(nom, "w");
    // if (f == NULL) ...

    sscanf(argv[1], "%d", &N);

    for(int i=1; i<=N; i++) {
        fprintf(f, "%d\n", i);
    }
    fclose(f);
    return 0;
}
```

Et on vérifie que l'argument fourni est bien un entier

```
int main(int argc, char *argv[]) {
    char nom[128] = "toto.txt";
    FILE *f; int N;

    f = fopen(nom, "w");
    // if (f == NULL) ...

    if (sscanf(argv[1], "%d", &N) == 0) {
        perror(argv[1]);
        exit(1);
    }

    // etc.
```

Exemple : copie (lecture et écriture dans 2 fichiers)

```
int main(int argc, char *argv[]) {
    FILE *src, *dest; char c;

    if (argc < 3) { exit(1); }

    src = fopen(argv[1], "r");
    dest = fopen(argv[2], "w");
    if (src == NULL || dest == NULL) { exit(2); }

    fscanf(src, "%c", &c);
    while (!feof(src)) {
        fprintf(dest, "%c", c);
        fscanf(src, "%c", &c);
    }
    fclose(src); fclose(dest);
    return 0;
}
```