Rappels et compléments de C

Cyril Rabat
cyril.rabat@univ-reims.fr

Licence 3 Informatique - Info0601 - Systèmes d'exploitation - concepts avancés

2020-2021

Cours n°4

Rappels de C

Compléments

Version 4 janvier 2021

Table des matières

Descripteurs de fichier

- Associés à différents types de ressources :
  ↪ Fichiers
  ↪ Tubes
  ↪ Sockets...
- Attention cependant aux propriétés associées au descripteur :
  ↪ Dépend de l'ouverture (options spécifiées)
  ↪ Des paramètres par défaut (selon les ressources associées)
- À noter qu'il est possible de modifier les propriétés avec `fcntl`

Écriture et lecture

- Utilisation des appels système `write` et `read`
- Données copiées bit-à-bit :
  ↪ Pas d'interprétation !
  ↪ Pointeurs génériques (`void*`)
- Paramètres correspondant aux données à lire/écrire :
  ↪ Adresse mémoire (pointeur) + taille des données
- Attention cependant à la représentation des données :
  ↪ Pas de problème entre processus locaux
  ↪ Problèmes lors de lecture/écriture sur différents hôtes
  ↪ Problèmes d'architecture, de systèmes, etc.

Utilisation des pointeurs génériques

- Peuvent représenter tout type de donnée (ou structure)
- Pour accéder aux données, transtypage (*cast*)
- Attention à l'ordre :
  - Soit : `void *ptr`
  - `(int)(*ptr)` → Interdit !
    ↪ `deferencing 'void *' pointer`
  - `*(int*)*ptr` → Autorisé

Taille des données : sizeof

- sizeof permet de retourner la taille des données en octets
- Important : ne pas spécifier la taille directement dans le code !
  ↪ Portable (suivant l'architecture, le système...)
  ↪ Évite les erreurs !

Exemples

- Avec int i :
  ↪ sizeof(i) = int = 4o
- Avec int t[10] :
  ↪ sizeof(t) = int[10] = 10×4o

Cas des pointeurs

- Tailles différentes en 32 bits (4o) et 64 bits (8o)
- Éviter les confusions entre pointeur et données pointées

Exemples

- Avec `int *i` :
  ↪ `sizeof(i)` = `int*` = 8o (sur 64 bits)
  ↪ Taille indépendante de l'initialisation de `i`
- Attention à la taille de ce qui est pointé :
  ↪ `sizeof(*i)` = 4o

Résumé sur les écritures/lectures

- Types primitifs : `write(fd, adresse, sizeof(type))`
  - `fd` : le descripteur de fichier
  - `adresse` : pointeur vers les données
  - `type` : le type des données pointées
- Tableau de type primitif : `write(fd, tab, sizeof(type)*taille)`
  - `fd` : le descripteur de fichier
  - `tab` : pointeur vers les données (les cases)
  - `type` : le type des données de chaque case
  - `taille` : le nombre de cases du tableau

Important

Pour la taille des données, utilisez `sizeof`

Les chaînes de caractères en C

- Source de nombreux *segmentation fault* :
  ↪ Accès mémoires interdits
- Erreurs courantes :
  - Espace mémoire non alloué (avec `char*`)
  - Dépassement de la capacité allouée
  - Mauvaise maîtrise des fonctions de la bibliothèque (`string.h`)
  - Problème du caractère de fin '\0'
  - Confusions `char*` et `char[]`

Utilisation de scanf pour les chaînes de caractères

- scanf avec %s :
  ↪ Spécification obligatoire de la longueur maximale
- Exemple : scanf("%10s", s)
  ↪ Attention : la chaîne doit être allouée et de taille 11 (pour le '\0')
- Pour la gestion des espaces :
  - scanf("%10[A-Z]", s) :
    ↪ 10 lettres majuscules maximum + espace
  - scanf("%10[^\n]", s) :
    ↪ Tout sauf le retour à la ligne

Comment lire plusieurs chaînes ?

- Avec `%s`, lecture jusqu'au délimiteur
- Ce dernier reste dans le tampon !
- Rappel : ne pas utiliser `fflush` !
  ↪ Possible uniquement sur les flux en sortie
- Possibilité : lire tous les caractères restants un par un

Vider le tampon d'entrée

```
char c;

while(((c = getchar()) != '\n') || (c == EOF));
```

Exemple de code complet

```c
#include <stdio.h>    /* Pour exit, EXIT_FAILURE, EXIT_SUCCESS */
#include <stdlib.h>   /* Pour printf, scanf, getchar, perror */

int main() {
  char s1[10];
  char s2[10];
  char c;

  printf("Saisir votre nom : ");
  if(scanf("%9s", s1) == EOF) {
    perror("Erreur du scanf "); exit(EXIT_FAILURE);
  }
  while(((c = getchar()) != '\n') || (c == EOF));

  printf("Saisir votre prenom : ");
  if(scanf("%9s", s2) == EOF) {
    perror("Erreur du scanf "); exit(EXIT_FAILURE);
  }
  while(((c = getchar()) != '\n') || (c == EOF));

  return EXIT_SUCCESS;
}
```

Utilisation de gets et fgets

- gets et fgets permettent de lire des chaînes de caractères :
  ↪ Utilisation du délimiteur retour à la ligne ou EOF
  ↪ Simplifie la lecture
- Mais ne pas utiliser gets : fonction dépréciée !
  ↪ Impossible de fixer une taille maximale !
- Sur l'utilisation de fgets :
  - La taille du buffer est spécifiée
  - Contrairement à scanf : le '\0' est compris dans la taille
  - Le délimiteur est lu et stocké dans la chaîne

Premier exemple

```
int main() {
  char buffer1[16] = "Bonjour";
  char buffer2[16] = "Au␣revoir";

  printf("Chaines␣:␣%s␣et␣%s\n", buffer1, buffer2);

  return EXIT_SUCCESS;
}
```

- Code correct :
  ↪ '\0' ajouté à la compilation

Deuxième exemple

```
int main() {
  char buffer1[16] = "abcdefghijklmnop";
  char buffer2[16] = "abcdefghijklmnop";

  printf("Chaines␣:␣%s␣et␣%s\n", buffer1, buffer2);

  return EXIT_SUCCESS;
}
```

- Affichage :

  ↪ `Chaines : abcdefghijklmnopabcdefghijklmnop et ab`

- '\0' non ajouté par manque de place !

Pour conclure

- Possibilité d'initialiser les chaînes de manière statique
- Le '\0' est ajouté automatiquement
- Attention à la taille déclarée : elle doit tenir compte du '\0'

Stockage de chaînes dans un fichier

- Exemple : `char str[10] = "toto\0";`

1. Doit-on stocker tous les caractères alloués ?
   ↪ Utilisation de `sizeof(char) × 10`
   ↪ Stocké : `toto ?????` (10 × `sizeof(char)` octets)
2. Uniquement les caractères utiles (avant le '\0') ?
   ↪ Utilisation de `strlen(str)`
   ↪ Stocké : `toto` (4 × `sizeof(char)` octets)
3. Les caractères + le \0 ?
   ↪ Utilisation de `strlen(str) + 1`
   ↪ Stocké : `toto` (5 × `sizeof(char)` octets)

Exemples de codes (sans gestion d'erreur)

```c
/* Écriture */
char str[10] = "Cool";
int fd;

fd = open("toto.bin", O_WRONLY|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR);

write(fd, str, sizeof(char) * 10);

close(fd);
```

```c
/* Lecture */
char str[10];
int fd;

fd = open("toto.bin", O_RDONLY, S_IRUSR|S_IWUSR);

read(fd, str, sizeof(char) * 10);
printf("Lu: '%s'\n", str);

close(fd);
```