Systèmes d'exploitation - TD/TP 3
Entrées-Sorties de base

Michel Meynard

14 janvier 2014

1 Entrées-Sorties fichiers

Il s'agit de se familiariser avec les différentes possibilités de faire des entrées-sorties en C. On va s'efforcer tout d'abord de faire quelques manipulations simples sur les fichiers. Cette mise au point permettra de passer à la gestion des fichiers dans le cours.

Exercice 1 (TD)
Quels sont les fichiers ouverts lors du lancement de tout processus ? Comment les changer depuis la ligne de commande ?

Exercice 2 (TD)
Quels sont les moyens que vous connaissez pour créer des fichiers dits fichier *texte* (contenant les caractères ASCII compris entre 20_{16} et $7F_{16}$).

Exercice 3 (TD)
Qu'est-ce qui distingue un fichier *texte* d'un fichier *binaire* ?

2 Appels systèmes

Exercice 4 (TD)
Connaissez-vous les appels systèmes de base suivantes de manipulation des fichiers :
- ouverture, fermeture,
- lecture, écriture,
- positionnement,
- test de fin de fichier ?

Exercice 5 (TD/TP Comptage des caractères différents)
On souhaite connaître le nombre de caractères différents présents dans un fichier. Par exemple, Si le fichier `toto.txt` possède le contenu suivant : `bbbabba`

Alors, le programme `compte` devra afficher ce qui suit :

```
compte toto.txt
2 caractères différents : a, b,
```

De même, Si le fichier `titi.txt` possède le contenu suivant : `allo olla`

Alors, le programme `compte` devra afficher ce qui suit :

```
compte titi.txt
4 caractères différents :  , a, l, o,
```

Le codage du fichier est un codage où chaque caractère est codé sur 1 octet (ISO-Latin1) et l'ordre d'affichage des caractères **n'a aucune importance**.

Questions

1. Expliquer **clairement** la ou les structures de données que vous comptez utiliser pour mémoriser le nombre de caractère : schéma + définition C ou C++.
2. Ecrire l'algorithme réalisant ce comptage.
3. Ecrire le programme C compte.c ;

Exercice 6 (TD/TP Nombre d'occurrences de caractères)
Certains algorithme de compression (Huffman) nécessitent de connaître le nombre d'apparition de chaque caractère présent dans un fichier. Par exemple, Si le fichier `toto.txt` possède le contenu suivant :

```
Le corbeau et le renard
Maître corbeau
```

Alors, le programme qu'on cherche à développer devra afficher ce qui suit :

```
occurrences toto.txt
L:1  e:7   :5  c:2  o:2  r:5  b:2  a:4  u:2  t:2  l:1  n:1  d:1
:1  M:1  î:1
```

En effet, ce fichier contient 7 lettres "e", 4 "a", ... Le codage du fichier est un codage où chaque caractère est codé sur 1 octet (ISO-Latin1)

Questions

1. Pourquoi la deuxième ligne de l'affichage est décalé d'un cran ?
2. Expliquer la ou les structures de données que vous comptez utiliser pour mémoriser le nombre de chaque caractère.
3. Ecrire l'algorithme réalisant ce comptage d'occurrences.
4. Ecrire le programme C `occurrences.c` ;

Exercice 7 (TD/TP hexl)
La commande `hexl` permet de visualiser un fichier sous sa forme hexadécimale (dump hexa) et sous sa forme texte. Chaque ligne du dump est composé de la position en hexadécimal, de 16 codes hexa, de 16 caractères affichables. L'exemple suivant illustre le propos :

```
hexl compte.c
00000000: 2369 6e63 6c75 6465 203c 7374 6469 6f2e  #include <stdio.
00000010: 683e 0909 0d0a 2369 6e63 6c75 6465 203c  h>....#include <
00000020: 7379 732f 7479 7065 732e 683e 0909 2f2a  sys/types.h>../*
00000030: 206f 7065 6e20 2a2f 0d0a 2369 6e63 6c75   open */..#inclu
```

1. Ecrire l'algorithme ;
2. Ecrire le programme correspondant.

Exercice 8 (TD/TP Écriture sur un fichier)
On souhaite réécrire la commande `cp source dest` qui permet de copier le fichier source dans le fichier destination.

Questions

1. Ecrire l'algorithme.
2. Ecrire le programme C `moncp.c` ;

Exercice 9 (TD/TP Positionnement dans un fichier)
Soit un fichier contenant des caractères 8 bits uniques et triés dans l'ordre croissant de leur code ASCII comme dans l'exemple suivant :

```
15678ACFGJKJLMXYZabcduvw
```

On souhaite tester la présence d'un char dans ce fichier en faisant une recherche dichotomique (on divise l'espace en deux à chaque pas). Par exemple :

```
dicho fic.txt 8
Le caractère 8 est en position 4
```

Questions

1. Comment connaître la taille du fichier ?
2. Ecrire l'algorithme.
3. Ecrire le programme C `dicho.c` ;

3 Fonctions de bibliothèque

Exercice 10 (TD)
Connaissez-vous les appels de fonctions de bibliothèque C suivantes de manipulation des fichiers :
- ouverture, fermeture,
- lecture, écriture,
- positionnement,
- test de fin de fichier ?

Exercice 11 (TD/TP)
L'objectif est de comparer ce qui se passe lorsqu'on utilise des appels système et les fonctions de bibliothèque.

1. Quels sont les avantages et inconvénients d'utiliser l'une ou l'autre approche ?
2. On peut (doit) réécrire certains exercices en utilisant ces fonctions de bibliothèque.

Solutions des exercices du TD/TP 3

Solution 1 Au lancement de tout processus, il y a 3 fichiers ouverts, numérotés 0,1 et 2, représentant respectivement *stdin*, i.e. l'entrée standard, *stdout* i.e. la sortie standard et stderr, i.e. l'erreur standard. Au lancement, les trois sont affectés au «terminal» (i.e. fenêtre dans laquelle a été lancé le processus), **sauf** s'ils sont redirigés.

Concernant les redirections, on peut lancer un processus en mettant par exemple :

```
monprog <metro >boulot 2>dodo
```

`<metro` : les lectures faites sur l'entrée standard se feront en fait sur le fichier `metro`,
`>boulot` : les écritures sur la sortie standard se font sur le fichier `boulot`) et
`2>dodo` : les écritures sur l'erreur standard se font sur le fichier `dodo`; (syntaxe bash)

Solution 2 édition, copie, commande `cat >fichu` (copie du clavier vers le fichier avec ctrl-D qui sert de fin de fichier), commande `touch` (c'est un effet de bord de cette commande)

Solution 3 Cela dépend de la destination du fichier. Un fichier *texte* est lisible «humainement» et contient principalement des caractères ASCII compris entre 20_{16} et $7F_{16}$ (sauf en ISO-Latin-1 ou en URF-8 !). Un fichier *binaire* contient des données codées selon une forme non lisible «humainement». Exemples : un exécutable, mais aussi un fichier de données contenant des entiers codés `int`. Il faut distinguer selon le codage un entier lisible (format chaîne de caractères) et non lisible (format interne en Complément à 2).

Solution 4
- open, close
- read, write
- lseek
- pas d'appel mais vrai lorsque un read ne lit pas le nombre de caractères demandés ;

Solution 5

1. Un tableau de 256 entiers (booléens) avec 1 case par caractère (0 absent, 1 présent). Un compteur entier.
2. **Algorithme**

```
f=ouvrir(argv[1])
si (f==-1)
  ret 1
int present[256] initialisé à 0
int compte=0
char c
Tantque (0<lire(f,c))
  Si non present[c]
    present[c]=1
    compte++
fermer(f)
afficher compte "caractères différents : "
Pour c=0 à 255
  Si present[c]
    afficher c ", "
```

3. **compte.c**

```
#include <stdio.h>
#include <sys/types.h>/* open */
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>/* read */


int main(int argc, char** argv, char** arge){
  if(argc!=2){
    printf("Syntaxe incorrecte : %s fichier.txt\n", argv[0]);
    return 1;
  }
  int f=open(argv[1],O_RDONLY);
  if (f<0){
    printf("Impossible d'ouvrir  %s !\n", argv[1]);
    return 2;
  }
  int present[256];
```

```
  for(int i=0;i<256;i++) /* pas de char i sinon boucle ! */
    present[i]=0;
  int compte=0;
  char c;
  while(0<read(f,&c,1)){
    if(!present[(int)c]){
      present[(int)c]=1;
      compte++;
    }
  }
  close(f);
  printf("%d caractères différents : ", compte);
  for(int i=0;i<256;i++)
    if(present[i])
      printf("%c, ",i);
  printf("\n");
}
```

Solution 6

1. Parce que le caractère affiché est un retour ligne.
2. Un tableau de 256 entiers avec 1 case par caractère comptant le nombre d'occurrences ;
3. **Algorithme**

```
f=ouvrir(argv[1])
si (f==-1)
  ret 1
int occ[256] initialisé à 0
char c
Tantque (0<lire(f,c))
  occ[c]++
fermer(f)
Pour c=0 à 255
  Si occ[c]
    afficher c ":" occ[c] ", "
```

4. **occurrences.c**

```
#include <stdio.h>
#include <sys/types.h>/* open */
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>/* read */


int main(int argc, char** argv, char** arge){
  if(argc!=2){
    printf("Syntaxe incorrecte : %s fichier.txt\n", argv[0]);
    return 1;
  }
  int f=open(argv[1],O_RDONLY);
  if (f<0){
    printf("Impossible d'ouvrir  %s !\n", argv[1]);
    return 2;
  }
  int occ[256];
  for(int i=0;i<256;i++) /* pas de char i sinon boucle ! */
    occ[i]=0;
  char c;
  while(0<read(f,&c,1)){
    occ[(int)c]++;
  }
  close(f);
  for(int i=0;i<256;i++)
    if(occ[i])
      printf("%c(0x%x):%d, ",i,i,occ[i]);
  printf("\n");
}
```

Solution 7

1. `f=ouvrir(argv[1])`

```
si (f==-1)
  ret 1
char c[16]
int position=0
int nb
Tantque (0<nb=lire(f,c,16))
  afficherHexa position ": "
  position=position+16
  Pour i=0 à nb-1
    afficherHexa c[i]
    Si i%2
      afficher " "
  Si nb<16
    Pour i=nb à 15
      afficher "  "
      Si i%2
        afficher " "
  afficher c "\n"
fermer(f)
```

2. **monhexl.c**

```
#include <stdio.h>
#include <sys/types.h>/* open */
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>/* read */


int main(int argc, char** argv, char** arge){
  if(argc!=2){
    printf("Syntaxe incorrecte : %s fichier.txt\n", argv[0]);
    return 1;
  }
  int f=open(argv[1],O_RDONLY);
  if (f<0){
    printf("Impossible d'ouvrir  %s !\n", argv[1]);
    return 2;
  }
  char c[16];
  int position=0,nb;
  while(0<(nb=read(f,&c,16))){
    printf("%8.8x: ",position); /* position sur 8 car en hexa */
    position+=16;
    for(int i=0;i<nb;i++){
      char code=c[i];
      printf("%2.2hhx",code); // hh pour indiquer que c'est un char (pas int)
      if(i%2) /* tous les 2 car, afficher un espace */
printf(" ");
    }
    if(nb<16){ /* pour la dernière ligne, compléter */
      for(int i=nb;i<16;i++){
printf("  ");
if(i%2) /* tous les 2 car, afficher un espace */
  printf(" ");
      }
    }
    printf(" ");
    for(int i=0;i<nb;i++){ /* format caractères */
      if(c[i]>=0x20 && c[i]<0x7F) /* ASCII affichable */
printf("%c",c[i]);
      else
printf("."); /* non affichable */
    }
```

```
    printf("\n");
  }
  close(f);
}
```

Solution 8 1. **Algorithme**

```
source=ouvrir(argv[1], lecture)
si (source==-1)
  ret 1
dest=ouvrir(argv[1], écriture)
si (dest==-1)
  ret 2
Tantque (0<lire(source, c))
  écrire(dest, c)
fermer(dest)
fermer(source)
```

2. **moncp.c**

```
#include <stdio.h>
#include <sys/types.h>/* open */
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>/* read */


int main(int argc, char** argv, char** arge){
  if(argc!=3){
    printf("Syntaxe incorrecte : %s source.txt destination.txt\n", argv[0]);
    return 1;
  }
  int source=open(argv[1],O_RDONLY);
  if (source<0){
    printf("Impossible d'ouvrir  %s !\n", argv[1]);
    return 2;
  }
  int dest=open(argv[2],O_WRONLY|O_CREAT|O_TRUNC,0x644);
  if (dest<0){
    printf("Impossible d'ouvrir  %s !\n", argv[2]);
    return 3;
  }
  char c;
  while(0<read(source,&c,1)){
    write(dest,&c,1);
  }
  close(dest);
  close(source);
}
```

Solution 9 1. `lseek` retourne la position du pointeur, il suffit donc de positionner celui-ci en fin de fichier `SEEK_END` pour connaître sa taille.

2. **Algorithme**

```
f=ouvrir(argv[1], lecture)
si (f==-1)
  ret 1
char x=premier(argv[2])
int debut=0;
int fin=lseek(f,0,SEEK_END) -1
si fin==-1
  ret2
repeter
  int milieu=(debut+fin)/2
  lseek(f,milieu,SEEK_SET)       // se placer au milieu
  lire(f, c)
  si c>x
```

```
    fin=milieu-1
  sinon si c<x
    debut=milieu+1
tantQue c!=x et debut<=fin
si c==x
  afficher "Le caractère " c " est en position " milieu
sinon
  afficher "Le caractère " c " est introuvable !"
fermer(f)
```

3. **dicho.c**

```
#include <stdio.h>
#include <sys/types.h>/* open */
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>/* read */
#include <string.h>/* strlen */

int main(int argc, char** argv, char** arge){
  if(argc!=3 || strlen(argv[2])!=1){
    printf("Syntaxe incorrecte : %s fic.txt c\n", argv[0]);
    return 1;
  }
  int f=open(argv[1],O_RDONLY);
  if (f<0){
    printf("Impossible d'ouvrir  %s !\n", argv[1]);
    return 2;
  }
  char x=argv[2][0]; //printf("x=%c; ",x);
  int debut=0;
  int fin=lseek(f,0,SEEK_END)-1; //printf("fin:%d; ",fin);
  if (fin==-1){
    printf("Impossible de se déplacer dans  %s !\n", argv[1]);
    return 3;
  }else if (fin==0){
    printf("Fichier vide : %s !\n", argv[1]);
    return 4;
  }
  int milieu,i;
  char c;
  do {
    milieu=(debut+fin)/2;       //printf("milieu=%d; ",milieu);
    i=lseek(f,milieu,SEEK_SET);       // se placer au milieu
    if (i==-1){
      printf("Impossible de se déplacer en %d dans %s !\n",milieu, argv[1]);
      return 5;
    }
    i=read(f,&c,1); //printf("c=%c; ",c);
    if (i!=1){
      printf("Impossible de lire dans %s !\n", argv[1]);
      return 6;
    }
    if(c>x){
      fin=milieu-1;
    }else if(c<x){
      debut=milieu+1;
    }
  }while(c!=x && debut<=fin);
  close(f);
  if(c==x)
    printf("Le caractère %c est en position %d !\n",x, milieu);
  else
    printf("Le caractère %c est introuvable !\n",x);
}
```

Solution 10
- fopen, fclose
- fgetc (un char), fgets (une ligne), fputc, fputs
- fscanf (lecture formattée), fprintf (écriture formattée)
- fseek
- vrai lorsque le FILE* == EOF

Solution 11
1. Les appels systèmes sont plus rapides, moins gourmands en mémoire donc plus efficaces. Mais ils sont plus simples : pas de lecture ni d'écriture formattée ! De plus, ils sont moins portables : d'un système Unix à un autre, il peut y avoir quelques différences. Ces problèmes de portabilité sont de moins en moins nombreux avec la conformité au standards tels que POSIX.