INFORMATIQUE
TECHNIQUE

Expert

3ième édition

# C en action

Téléchargement
www.editions-eni.fr

Yves METTIER

eni Editions

Les éléments à télécharger sont disponibles à l'adresse suivante :
**http://www.editions-eni.fr**
Saisissez la référence de l'ouvrage **EI3CACT** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

## Introduction



## Chapitre 1
## Bases du langage C

## Chapitre 6
## Gestion des erreurs



## Chapitre 7
## Structures de données

## Chapitre 8
## Dates et heures



## Chapitre 9
## Chaînes de caractères

**Chapitre 12**
## Contenu des fichiers

**Chapitre 13**
## Réseau

## Chapitre 14
## Protocoles réseau



## Chapitre 15
## Signaux

**Chapitre 16**
## Exécution parallèle



**Chapitre 17**
## Système et processus

**Chapitre 23**
## C11 et les normes du langage C



# Chapitre 11
# Gestion des répertoires et des fichiers

## 1. Introduction

Sur les systèmes Unix, il est souvent dit que tout est fichier. Cela s'applique notamment aux répertoires. Et les fonctions utilisables pour les fichiers le sont aussi pour les répertoires. Cependant, les répertoires étant des fichiers particuliers puisque le contenu n'est rien d'autre que la liste des fichiers qu'ils contiennent, nous disposons de quelques fonctions supplémentaires qui nous permettent de les ouvrir, de lire leur contenu entrée par entrée, de les créer simplement... Ce chapitre traitera des répertoires, ainsi que de leur contenu, les fichiers. Vous trouverez donc le nécessaire pour agir sur ceux-ci, à l'exception du contenu des fichiers qui fait l'objet du chapitre suivant.

## 2. Connaître le contenu d'un répertoire

### Problème

Vous voulez connaître les noms des fichiers et des répertoires contenus dans un répertoire.

### Solution

Servez-vous des fonctions `opendir()`, `readdir()` et `closedir()`. Utilisez-les de manière récursive dans les sous-répertoires si vous voulez aussi les parcourir.

### Discussion

L'utilisation d'`opendir()` est simple. La difficulté réside dans ce que nous souhaitons faire des noms des fichiers et des répertoires obtenus avec `readdir()`. Voici un cas d'école, un exemple qui affiche les noms des fichiers et sous-répertoires d'un répertoire :

```
int
printdir (char *dirname)
{
  DIR *FD;
  struct dirent *f;

  if (NULL == (FD = opendir (dirname)))
    {
      fprintf (stderr, "opendir() impossible\n");
      return (-1);
    }
  printf ("%s :\n", dirname);
  while ((f = readdir (FD)))
    {
      printf ("  %s\n", f->d_name);
    }
  closedir (FD);
  return (0);
}
```

Dans la boucle qui affiche les fichiers et répertoires, recourrez à la recette "Obtenir des informations sur un fichier" pour afficher plus que le nom des fichiers, en particulier s'il s'agit d'un fichier ou d'un répertoire. Cela permet, pour parcourir le répertoire de manière récursive, de réitérer le processus sur les répertoires ainsi rencontrés. La recette suivante applique cette méthode pour effectuer une opération récursivement sur tous les fichiers d'un répertoire et de ses sous-répertoires. S'il s'agit d'afficher le contenu d'un répertoire et de ses sous-répertoires, faites appel à cette recette, l'action étant d'afficher le nom du fichier.

**Prototypes**

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

DIR *opendir (const char *filename);

struct dirent *readdir (DIR * dirp);

int closedir (DIR * dirp);
```

Remarque

*Voir aussi les pages de manuel de `opendir()`, `readdir()` et `closedir()`.*

## 3. Effectuer une opération récursivement sur tous les fichiers d'un répertoire et de ses sous-répertoires

### Problème

Vous souhaitez effectuer une opération sur tous les fichiers d'un répertoire ainsi que ses sous-répertoires.

### Solution

Il n'y a pas de solution toute faite, vous devez programmer un algorithme de parcours des répertoires de manière récursive.

### Discussion

Il existe plusieurs manières de parcourir les répertoires. La méthode la plus simple consiste à tester chaque fichier pour savoir si c'est un répertoire, et le cas échéant, appliquer l'algorithme à ce répertoire, de manière récursive. Cette méthode a l'avantage d'être simple mais pose un problème d'ordre de parcours. En effet, souhaitez-vous parcourir tous les fichiers et répertoires d'un même niveau ou pouvez-vous vous contenter de traiter les sous-répertoires d'un niveau au moment où ils se présentent, et ne continuer le niveau en cours que plus tard ?

Le programme suivant montre une méthode proche de celle de la commande `find` sur GNU/Linux, à base de récursivité et en utilisant `chdir()` qui nous place en permanence dans le répertoire en cours de traitement. Cette méthode utilise une liste chaînée pour obtenir la liste des fichiers d'un répertoire. Cela permet de faire un classement optionnel avant traitement. Dans notre cas, le classement consiste à traiter d'abord les fichiers, puis à aller dans les sous-répertoires de manière récursive.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <errno.h>

/*****************************************
 * Action sur les fichiers et répertoires *
 *****************************************/

/* Effectue une action sur un répertoire. */
void
action_dir (const char *dir)
{
  printf ("%s/\n", dir);
}

/* Effectue une action sur un répertoire avant parcours récursif
 * de son contenu.
 */
void
action_dir_pre (const char *root, const char *dir)
{
  printf ("\n-> %s/%s/\n", root, dir);
}

/* Effectue une action sur un répertoire après parcours récursif
 * de ce répertoire.
 */
void
action_dir_post (const char *root, const char *dir)
{
}
```

```
/* Effectue une action sur un fichier. */
void
action_file (const char *file)
{
  printf ("%s\n", file);
}

/*************************************
 * Définition d'une liste chaînée simple *
 *************************************/

typedef struct slist_t
{
  char *name;
  int is_dir;
  struct slist_t *next;
} slist_t;

/***********************************
 * Parcours récursif des répertoires *
 ***********************************/

int
recursive_dir (char *root, char *dirname)
{
  slist_t *names = NULL;
  slist_t *sl;

  DIR *FD;
  struct dirent *f;
  int cwdlen = 32;
  char *cwd;
  char *new_root;

  if (NULL == (cwd = malloc (cwdlen * sizeof *cwd)))
    {
      fprintf (stderr, "Problème avec malloc\n");
      exit (EXIT_FAILURE);
    }

  /* Concaténation new_root = "root/dirname" */
  if (root)
    {
      int rootlen = strlen (root);
      int dirnamelen = strlen (dirname);
      if (NULL ==
```

```
        (new_root =
         malloc ((rootlen + dirnamelen + 2) * sizeof *new_root)))
      {
        fprintf (stderr, "Problème avec malloc\n");
        exit (EXIT_FAILURE);
      }
    memcpy (new_root, root, rootlen);
    new_root[rootlen] = '/';
    memcpy (new_root + rootlen + 1, dirname, dirnamelen);
    new_root[rootlen + dirnamelen + 1] = '\0';
  }
else
  new_root = strdup (dirname);

/* Obtention du répertoire courant */
while (NULL == (cwd = getcwd (cwd, cwdlen)))
  {
    if (ERANGE != errno)
      {
        fprintf (stderr, "Problème avec getcwd (errno = '%s')\n",
                 strerror (errno));
        exit (EXIT_FAILURE);
      }
    cwdlen += 32;
    cwd = realloc (cwd, cwdlen * sizeof *cwd);
  }
chdir (dirname);

/* Remplissage de la liste chaînée avec les noms des fichiers
 * du répertoire courant. */
if (NULL == (FD = opendir (".")))
  {
    fprintf (stderr, "opendir() impossible\n");
    return (-1);
  }
sl = names;
while ((f = readdir (FD)))
  {
    struct stat st;
    slist_t *n;
    if (!strcmp (f->d_name, "."))
      continue;
    if (!strcmp (f->d_name, ".."))
      continue;
    if (stat (f->d_name, &st))
      continue;
```