IR209 - Développement

Rapport de Laboratoire 3: Parcours récursif (suite & fin), Gestion des arguments & Allocation dynamique

Maljean Thimothé, Tasiaux Victor

19-10-2022

Table des Matières

1 Parcours récursif suite et fin

```c
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


void dirContent(const char rootpath);


int main(void){

    dirContent("/home/kali/Documents/Developpement/Lab2/exercice2/test"
        );

    return 0;
}

void dirContent(const charrootpath){
    puts(rootpath);
    DIR d = opendir(rootpath);
    struct direntsd = readdir(d);

    while (sd != NULL){

        if(sd -> d_type == DT_DIR){
         ### TO DO ###
        }

        else{
            if(sd -> d_type == DT_REG){
                printf("Nom du fichier: %s",sd->d_name);
            }
        }

    }




}
```

Ce code avait déjà été validé au cours précédent, il reste à créer un parcours de dossier récursif.

2 Gestion des arguments

2.1 Version 1

```c
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


void dirContent(const char *rootpath);


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

    dirContent(argv[1]);

    return 0;
}


void dirContent(const char *rootpath){
    puts(rootpath);
    DIR *d = opendir(rootpath);
    struct dirent *sd = readdir(d);

    if (d == NULL){
        perror("ERROR: ");
    }

    while (sd != NULL){

        if(sd -> d_type == DT_DIR){
        }

        else{
            if(sd -> d_type == DT_REG){
                printf("Nom du fichier: %s",sd->d_name);
            }
        }

    }

}
```

Dans cette version, on a vidé la partie lecture de directory pour se concentrer sur l'`argv`. On appelle la fonction `dirContent`, et on passe le string entré dans le terminal en paramètre de la fonction.

2.2 Version 2

```c
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


void dirContent(const char *rootpath);
```

```c

int main(int argc, char* argv[]){
    if(argc == 2){
        dirContent(argv[1]);
    }
    else{

        if(argc > 2){
            puts("ERROR: too many arguments");
            return 0;
        }

        else {
            puts("ERROR: too few arguments");
            return 0;
        }
    }

    return 0;
}


void dirContent(const char *rootpath){
    DIR *d = opendir(rootpath);

    if (d == NULL){
        perror("ERROR: ");
    }
    struct dirent *sd = readdir(d);



    while (sd!= NULL){

       if(sd -> d_type == DT_DIR){

        }


        else{
            if( sd -> d_type == DT_REG){
                printf("Nom du fichier: %s\n",sd->d_name);
            }
        }
       sd = readdir(d);
    }
}
```

Dans cette version on ajoute un contrôle du nombre d'arguments grâce à `argc`.

3 Allocation dynamique du Path

3.1 Version 1: tableau à taille définie

```c
#include <dirent.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>


void dirContent(const char *rootpath);


int main(int argc, char* argv[]){
    if(argc == 2){
        dirContent(argv[1]);
    }
    else{

        if(argc > 2){
            puts("ERROR: too many arguments");
            return 0;
        }

        else {
            puts("ERROR: too few arguments");
            return 0;
        }
    }

    return 0;
}


void dirContent(const char *rootpath){
    DIR *d = opendir(rootpath);

    if (d == NULL){
        perror("ERROR: ");
    }
    struct dirent *sd = readdir(d);



    while (sd!= NULL){

        if(sd -> d_type == DT_DIR && strcmp(sd -> d_name,".") !=0 &&
           strcmp(sd -> d_name,"..") !=0 ){
            char new_path[4096] = "";
            strncat(new_path,rootpath,strlen(rootpath));
            strncat(new_path,"/",1);
            strncat(new_path,sd ->d_name,strlen(sd->d_name));
            printf("Path:%s\n",new_path);

            dirContent(new_path);
        }

        else{
            if( sd -> d_type == DT_REG){
```

```
55                  printf("Nom du fichier: %s\n",sd->d_name);
56              }
57          }
58         sd = readdir(d);
59      }
60  }
```

Dans cette version, on utilise un tableau à taille fixe et `strncat`, version sécurisée de `strcat` pour créer le nouveau répertoire.

On vérifie que le fichier est un dossier et qu'il n'est pas le dossier courant(`.`) ou le dossier parent (`..`).

3.2 Version 2: Avec malloc

```
1  #include <dirent.h>
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <string.h>
5
6
7  void dirContent(const char *rootpath);
8
9
10 int main(int argc, char* argv[]){
11     if(argc == 2){
12         dirContent(argv[1]);
13     }
14     else{
15
16         if(argc > 2){
17             puts("ERROR: too many arguments");
18             return 0;
19         }
20
21         else {
22             puts("ERROR: too few arguments");
23             return 0;
24         }
25     }
26
27     return 0;
28 }
29
30
31 void dirContent(const char *rootpath){
32     DIR *d = opendir(rootpath);
33
34     if (d == NULL){
35         perror("ERROR: ");
36     }
37     struct dirent *sd = readdir(d);
38
39
40
41     while (sd!= NULL){
42
```

```
43          if(sd -> d_type == DT_DIR && strcmp(sd -> d_name,".") !=0 &&
                strcmp(sd -> d_name,"..") !=0 ){
44              char* new_path = (char*)malloc(strlen(rootpath)+strlen(sd->
                    d_name)+2);
45              strcpy(new_path,"");
46              strncat(new_path,rootpath,strlen(rootpath));
47              strcat(new_path,"/");
48              strncat(new_path,sd ->d_name,strlen(sd->d_name));
49              printf("Path:%s\n",new_path);
50
51
52              dirContent(new_path);
53              free(new_path);
54          }
55
56          else{
57              if( sd -> d_type == DT_REG){
58                  printf("Nom du fichier: %s\n",sd->d_name);
59              }
60          }
61         sd = readdir(d);
62      }
63  }
```

On emploie ici la fonction `malloc` pour créer un tableau de taille variable.

On alloue de la mémoire pour le nouveau chemin → c'est la longueur du `rootpath` + du `d_name` + 2. Le 2 correspond au "`/`" + le `\0` qui marque la fin du tableau. Comme `malloc` est à la base un **`void`**, on caste le malloc en tableau de char.