```
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define MAXPAROLA 30
#define MAXRIGA 80

int main(int argc, char *argv[])
{
   int freq[MAXPAROLA] ; /* vettore di contatori
   delle frequenze delle lunghezze delle parole */
   char riga[MAXRIGA] ;
   int i, inizio, lunghezza ;
   FILE * f ;

   for(i=0; i<MAXPAROLA; i++)
      freq[i]=0 ;

   if(argc != 2)
   {
      fprintf(stderr, "ERRORE, serve un parametro con il nome del file\n");
      exit(1);
   }
   f = fopen(argv[1], "rt") ;
   if(f==NULL)
   {
      fprintf(stderr, "ERRORE, impossibile aprire il file %s\n", argv[1]);
      exit(1);
   }

   while( fgets( riga, MAXRIGA, f ) != NULL )
```

System and Device Programming

The File System

Stefano Quer
Dipartimento di Automatica e Informatica
Politecnico di Torino

Directories

- No storage system contains a single file
- Files are organized in directories
  - A directory is a node (of a tree) or a vertex (of a graph) that stores information about the (regular) file that it contains
  - Both directories and files are saved in mass memory
- Operations that can be performed on directories are similar to the ones applied to files
  - Creation, deletion, listing, rename, visit, search, etc.

File system management

- The POSIX standard provides a set of functions to perform the manipulation of directories
  - The function **stat**
    - Allows to understand the type of "entry" (file, directory, link, etc.)
    - This operation is permitted using the C data structure returned by the function, i.e. **struct stat**
  - Some other functions to manage the file system
    - getcwd, chdir
    - mkdir, rmdir
    - opendir, readdir, closedir

Returned data structure

Positioning

Creation
Cancellation

Visit / Inspection

The function stat

Path to return information about

Returned data structure

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

int stat (const char *path, struct stat *sb);
int lstat (const char *path, struct stat *sb);
int fstat (int fd, struct stat *sb);
```

- The function **stat** returns a reference to the structure **sb** (**struct stat**) for the file (or file descriptor) passed as a parameter
- Return value
  - The value 0, on success
  - The value -1, on error

The function stat

- The function
  - **lstat** returns information about the symbolic link, not the file pointed by the link (when the path is referred to a link)
  - **fstat** returns information about a file already opened (it receives the file descriptor instead of a path)

```
int stat (const char *path, struct stat *sb);
int lstat (const char *path, struct stat *sb);
int fstat (int fd, struct stat *sb);
```

The function stat

```
struct stat {
  mode_t st_mode;       /* file type & mode */
  ino_t st_ino;         /* i-node number */
  dev_t st_dev;         /* device number */
  dev_t st_rdev;        /* device number */
  ...
};
```

- The second argument of **stat** is the pointer to the structure **stat**
- The field **st_mode** encodes the file type

The function stat

```
struct stat {
  mode_t st_mode;        /* file type & mode */
  ino_t st_ino;          /* i-node number */
  dev_t st_dev;          /* device number */
  dev_t st_rdev;         /* device number */
  ...
};
```

- Some macros allow to understand the type of the file
  - **S_ISREG** regular file, **S_ISDIR** directory, **S_ISBLK** block special file, **S_ISCHR** character special file, **S_ISFIFO** FIFO, **S_ISSOCK** socket, **S_ISLNK** symbolic link

Example

Check the directory entry type

Allow to understand if it is a directory !

```
struct stat buf;
...
if (lstat(argv[i], &buf) < 0) {
  fprintf (stdout, "lstat error.\n");
  exit(1);
}
if      (S_ISREG(buf.st_mode))  ptr = "regular";
else if (S_ISDIR(buf.st_mode))  ptr = "directory";
else if (S_ISCHR(buf.st_mode))  ptr = "char special";
else if (S_ISBLK(buf.st_mode))  ptr = "block special";
else if (S_ISFIFO(buf.st_mode)) ptr = "fifo";
else if (S_ISLNK(buf.st_mode))  ptr = "symbolic link";
else if (S_ISSOCK(buf.st_mode)) ptr = "socket";
     printf("%s\n", ptr);
}
```

Functions getcwd and chdir

```
#include <unistd.h>

char *getcwd (char *buf, int size);

int chdir (char *path);
```

Dimension of buf

Get Current Working Directory

Change Directory

- Get (change) the path of the **working directory**
- Returned values
  - getcwd
    - The buffer buf on success; NULL on error
  - chdir
    - 0 on success; -1 on error

Example

How to use getcwd and chdir

```
#define N 100

char name[N];

if (getcwd (name, N) == NULL)
  fprintf (stderr, "getcwd failed.\n");
else
  fprintf (stdout, "dir %s\n", name);

if (chdir(argv[1]) < 0)
  fprintf (stderr, "chdir failed.\n");
else
  fprintf (stdout, "dir changed to %s\n", argv[1]);
```

Functions mkdir and rmdir

See system call open

```
#include <unistd.h>
#include <sys/stat.h>

int mkdir (const char *path, mode_t mode);

int rmdir (const char *path);
```

- **mkdir** creates a new (empty) directory
- **rmdir** deletes a directory (if it is empty)
- Returned values
  - 0 on success
  - -1 on error

Functions opendir, dirent, closedir

```
#include <dirent.h>


DIR *opendir (
  const char *filename
);


struct dirent *readdir (
  DIR *dp
);


int closedir (
  DIR *dp
);
```

Open a directory for reading
Return value:
The pointer to the directory, on success
The NULL pointer, on error

Proceed with the reading of the directory. Return value:
The pointer to the directory, on success
The NULL pointer, on error or at the end of the reading operation

Terminate the reading
Return value:
0, on success
-1, on error

The structure dirent

```
struct dirent {
  inot_t d_no;
  char d_name[NAM_MAX+1];
  ...
}
```

- The structure **dirent** returned by **readdir**
  - Has a format that depends on the specific implementation
  - It contains at least the following fields
    - The i-node number
    - The file name (null-terminated)

Example

Visit a directory and print its content

```
#define N 100
...
struct stat buf;
DIR *dp;
char fullName[N];
struct dirent *dirp;
int i;
...
if (lstat(argv[1], &buf) < 0 ) {
  fprintf (stderr, "Error.\n"); exit (1);
}
if (S_ISDIR(buf.st_mode) == 0) {
  fprintf (stderr, "Error.\n"); exit (1);
}
if ( (dp = opendir(argv[1])) == NULL) {
  fprintf (stderr, "Error.\n"); exit (1);
}
```

Structure for lstat

Directory "handle"

Structure for readdir

Ask information about the path in argv[1]

If it is not a directory, the program terminates

Otherwise, the directory is open

Example

```
i = 0;
while ( (dirp = readdir(dp)) != NULL) {
  sprintf (fullName, "%s/%s", argv[1], dirp->d_name);
  if (lstat(fullName, &buf) < 0 ) {
    fprintf (stderr, "Error.\n"); exit (1);
  }
  if (S_ISDIR(buf.st_mode) == 0) {
    fprintf (stdout, "File %d: %s\n", i, fullName);
  } else {
    fprintf (stdout, "Dir  %d: %s\n", i, fullName);
  }
  i++;
}
if (closedir(dp) < 0) {
  fprintf (stderr, "Error.\n"); exit (1);
}
```

Read the directory (iterating over all entries)

Request information about the entry fullName

Display data

Closure and termination

Observations

- To visit one or more directory trees
  - The visit function must use recursion
  - Avoid recurring in subdirectory
    - "." the directory itslef
    - ".." directory parent
  - Manipulate the path correctly through string concatenation (sptrinf, strcat, etc.)
  - Keep into account that the current directory is a process-related infomation
    - All threads share the same current process directory