Maciej Sobieraj

Lecture 11

Outline

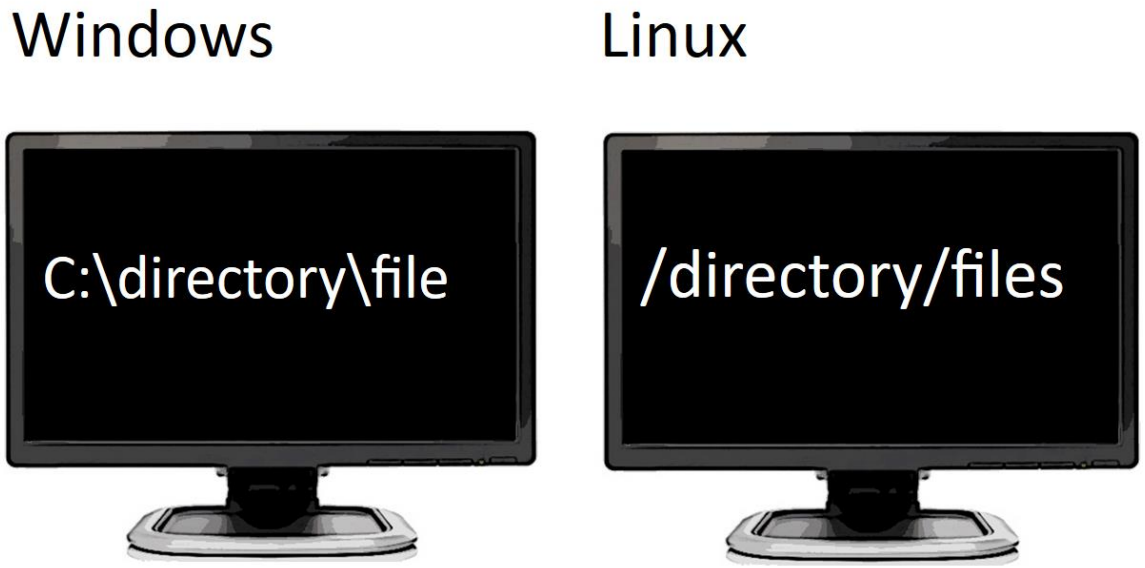
1. Connecting to the real world: files and streams
    1. **File systems: definitions and conventions**
    2. Introduction to files and streams
    3. Opening streams
    4. Pre-opened streams
    5. Closing the stream and error handling
2. Quiz

Files: containers for data

- One of the most common issues in a developer's job is to **process data stored in files**. Imagine a program that sorts 20 numbers and the user of this program enters these twenty numbers directly from the keyboard.
- Now imagine the same task when there are 20,000 numbers to be sorted – there's no user who is able to enter these numbers without a mistake.

Names of files

- **Different operating systems** can treat files **differently**. For example, Windows uses a different naming convention than Unix/Linux systems.



Names of files

- In addition, Unix system file names are **case sensitive**, while Windows systems store the cases of letters used in the file name, but **don't distinguish their cases at all.**
- This means that these two strings → describe two different files in Unix systems, but are the same name of just one file in Windows systems.

ThisIsTheNameOfTheFile

thisisthenameofthefile

Names of files

- Suppose that we're interested in a particular file located in the *dir* directory, named *file*.
- Suppose also that we want to assign a string containing the name of the file. In the Unix system it would look as follows

```
char *name = "/dir/file";
```

Names of files

- But if you try to code it for the Windows system:
  - **char** *name = "c:\dir\file";
- You should expect an unpleasant surprise: either the compiler will reproach you or the program will execute, surprisingly, as if the file name has been distorted in a strange way.

```
char *name = "c:\\dir\\file";
```

- The “C” language uses the “\” as an **escape character**

Outline

1. Connecting to the real world: files and streams
    1. File systems: definitions and conventions
    2. **Introduction to files and streams**
    3. Opening streams
    4. Pre-opened streams
    5. Closing the stream and error handling
2. Quiz

Stream: a file abstraction

- Any program written in the “C” language does not communicate with the files **directly**, but through an abstract entity called a **stream**.
- The programmer, having a rich set of functions defined mainly in the header file *stdio.h*, can perform certain operations on the stream which affect the real files using mechanisms **contained in the operating system kernel.**

Stream: a file abstraction

- Operations performed with the **abstract stream** reflect the activities related to the **physical file**.
- To **connect** (bind) the stream with the file, you need to perform an explicit operation.
- The operation of connecting the stream with a file is called **opening** the file, while disconnecting this link is named **closing** the file.

What can we do with a stream?

- The opening of the stream is associated with the file and should also declare the manner in which the stream will be processed. This declaration is called an **open mode**.
- There are two basic operations performed on the stream:
  - **read** from the stream: portions of data are **retrieved** from the file and placed in a memory area managed by the program (e.g. a variable);
  - **write** to the stream: portions of data from the memory (e.g. a variable) are **transferred** to a file.

How can we open a stream?

- There are three basic modes used to open the stream:
  - **read mode**: a stream opened in this mode allows reading operations only – trying to write to the stream will cause a runtime error;
  - **write mode**: a stream opened in this mode allows writing operations only – attempting to read the stream will cause a runtime error;
  - **update mode**: a stream opened in this mode allows both writing and reading.

How can we open a stream?

- The stream behaves almost like a **tape recorder**. When you read something from a stream, a virtual **head** moves over the stream according to the number of bytes transferred from the stream.

How do we represent a stream in the program?

- The “C” language standard assumes that every programming environment should guarantee the existence of a type named FILE, which is used to represent streams in the program.
- The *FILE* type is defined inside the *stdio.h* header file. Any program using a stream needs to include this file.

```
#include <stdio.h>
```

Binary streams vs text streams

- Due to the type of the stream's content, all streams are divided into **text** and **binary** streams.
- The former are **structured in lines**; that is, they contain typographical **characters** (letters, digits, punctuation, etc.) arranged in rows (lines).
- This file is written (or read) mostly **character by character** or **line by line**.

Binary streams vs text streams

- The library functions are responsible for reading and writing characters to and from the stream. It happens in the following way:
  - when the stream is **open**, it should be advised that the data in the associated file is be processed as **text**;
  - while reading/writing lines from/to the associated file, a process called the **translation of newline characters** occurs: when you read a line from the file, every pair of *\r\n* characters is replaced with a single *\n* character.

Outline

1. Connecting to the real world: files and streams
    1. File systems: definitions and conventions
    2. Introduction to files and streams
    3. **Opening streams**
    4. Pre-opened streams
    5. Closing the stream and error handling
2. Quiz

fopen() function

- The opening of the stream is performed by a function of the following prototype

```
FILE *fopen(char *filename, char *openmode);
```

  - the name of the function comes from the words "file open";
  - if the opening is successful, the function returns a pointer to a newly created variable of type FILE, otherwise it returns NULL, which can be easily used to validate the invocation;

fopen() function

```
FILE *fopen(char *filename, char *openmode);
```

- the first parameter of the function specifies the name of the file name to be associated with the stream. The name must be written according to the conventions applicable in a particular operating system;
- the second parameter specifies the open mode used for the stream. It is described by a sequence of characters and each of them has its own special meaning;
- the opening must be the **very first operation** performed on the stream

Open modes: r

- “r” open mode: read
  - the stream will be opened in “read” mode;
  - the file associated with the stream must exist and has to be readable, otherwise the fopen function fails.

r

read

Open modes: w

- “w” open mode: write
  - the stream will be opened in “write” mode;
  - the file associated with the stream doesn't need to exist; if it doesn't exist it will be created; if it exists it will be truncated to the length of zero (erased); if creation isn't possible (e.g. due to system permissions) the fopen function fails.

write

Open modes: a

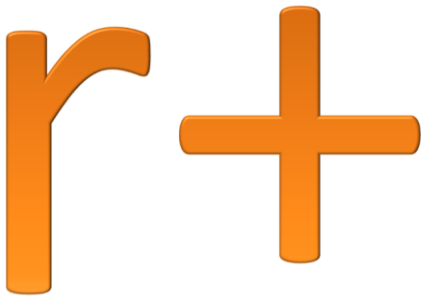
- The “a” open mode: **append**
  - the stream will be opened in “append” mode;
  - the file associated with the stream doesn't need to exist; if it doesn't exist, it will be created; if it does exist, the virtual recording head will be set to the end of the file (the previous content of the file remains untouched).

a

append

Open modes: r+

- The “r+” open mode: **read and update**
  - the stream will be opened in “read and update” mode;
  - the file associated with the stream must exist and has to be **writeable**, otherwise the *fopen* function fails;
  - both read and write operations are allowed for the stream.



Open modes: w+

- The “w+” open mode: **write and update**
  - the stream will be opened in “write and update” mode;
  - the file associated with the stream doesn't need to exist; if it doesn't exist it will be created; the previous content of the file is discarded (the file is truncated to zero length);
  - both read and write operations are allowed for the stream.

Specifying text and binary modes

- If the letter b is at the end of the mode string, it means that the stream is to be opened in binary mode. The default behavior when no binary/text mode specifier is used is to open the stream in text mode.
- Some compilers can specify that if the mode string ends with the letter **t**, the stream is opened in text mode. Also, opening a file in **text** mode is necessary for programs running on Windows systems.

Specifying text and binary modes

rt	rb
wt	wb
at	ab
r+t	r+b
w+t	w+b

Opening the stream: an example

- Imagine that we want to develop a program that reads the file named:
  - c:\users\user\Desktop\file.txt

```
FILE *file;

file = fopen("c:\\users\\user\\Desktop\\file.txt","rt");
if(file == NULL) {
    printf("File cannot be opened");
    return 1;
}
```

Outline

1. Connecting to the real world: files and streams
    1. File systems: definitions and conventions
    2. Introduction to files and streams
    3. Opening streams
    4. **Pre-opened streams**
    5. Closing the stream and error handling
2. Quiz

fopen() function

- Any stream operation **must be preceded** by the *fopen()* function invocation
- There are three well-defined exceptions to this rule. When our program starts, **three streams are already opened** and don't require any extra preparation.

```
FILE *stdin, *stdout, *stderr;
```

- Program can use these streams if it contains following directive:
  - #include <stdio.h>

stdin stream

- *stdin* (standard input);
- the *stdin* stream is normally associated with the **keyboard**, pre-opened for **reading** and regarded as the primary data source of running programs;
- the *scanf* function reads the data from *stdin* by default.

stdout stream

- *stdout* (standard output);
- the *stdout* stream is normally associated with the **screen**, pre-opened for **writing**, regarded as the primary target for outputting data by the running program;
- the *printf* function outputs the data to the *stdout* stream.

stderr stream

- *stderr* (standard error output);
- the *stderr* stream is normally associated with the **screen**, pre-opened for **writing**, regarded as the primary place where the running program should send information on the errors encountered during its work;
- we haven't shown you any function used to send the data to this stream yet (we'll do it soon, we promise).

Outline

1. Connecting to the real world: files and streams
    1. File systems: definitions and conventions
    2. Introduction to files and streams
    3. Opening streams
    4. Pre-opened streams
    5. **Closing the stream and error handling**
2. Quiz

fclose() function

- The last operation performed on a stream should be the **closing**. This action is performed by a function with the following prototype:

```
int fclose(FILE *stream);
```

- the function **returns 0 on success** or a value identified by the symbol *EOF* otherwise (the *EOF* symbol is declared in the *stdio.h* file and represents a value equal to -1; its name comes from the term *End Of File*).

fclose() function

- If we would like to close the stream opened in the previous example, we need to invoke the *fclose* function as follows:

```
fclose(file);
```

- Upon execution of *fclose*, the stream is no longer associated with any file, and any attempt to perform an operation on it (except *fopen*) will fail.

The errno variable

- The definition of the *errno* variable (the name comes from the phrase "*error number*") is located in the *errno.h* header file and it look as follows:

```
extern int errno;
```

- By definition, the execution of any function operating on a stream **sets the** *errno* **variable** with the error code identifying the reason for failure.

The errno variable

```
extern int errno;
```

- The value of the *errno* variable **can be compared** with one of the predefined symbolic constants (also defined in the *errno.h* file), which provide a basis for determining the actual reason for the error.

The errno variable

EACCES	Permission denied this error occurs when you try, for example, to open a file with the “read only” attribute for writing
EBADF	Bad file number this error occurs when you try, for example, to operate with an unopened stream
EEXIST	File exists This is an error that occurs when you try, for example, to rename a file with its previous name
EFBIG	File too large the error occurs when you create a file that is larger than the maximum allowed by the operating system
EISDIR	Is a directory this error occurs when you try to treat a directory name as an ordinary file name
EMFILE	Too many open files this error occurs when you try to simultaneously open more streams than acceptable by your operating system
ENOENT	No such file or directory this error occurs when you try to access a non-existent file/directory
ENOSPC	No space left on device this error occurs when there is no free space on the media

The errno variable

```
FILE *file = fopen("c:\\file.txt","rt");
if(file == NULL) {
    switch(errno) {
    case ENOENT: printf("The file doesn't exist\n");
                 break;
    case EMFILE: printf("You've opened too many files\n");
                 break;
    default:     printf("The error number is %d\n",errno);
    }
}
```

The errno variable

- Fortunately, there's a function that can dramatically **simplify** error-handling code.
- It's called *strerror*, its prototype is located in the *string.h* file.

```
char *strerror (int errnum);
```

- Now we can simplify our code in the following way:
  - FILE *file = fopen("c:\\file.txt","rt");
    if(file == NULL)
    printf("File could not be opened: %s\n",strerror(errno));

Outline

1. Connecting to the real world: files and streams
    1. File systems: definitions and conventions
    2. Introduction to files and streams
    3. Opening streams
    4. Pre-opened streams
    5. Closing the stream and error handling
2. **Quiz**

Quiz

If you open a file with the following `fopen` invocation:

```
FILE *f = fopen("f","w");
```

then you can:

- ○ update the stream
- ○ read from the stream
- ○ write to the stream

Quiz

Your program emits a message to the `stderr` stream. Where is the message shown?

- ○ nowhere
- ○ on the screen
- ○ in the file called stderr.log

Quiz

What should you do to be able to use `errno` for diagnostic purposes?

- declare a variable of that name
- include an appropriate header file
- declare a function of that name