Programming I

Lecture 10

Files

What did we talk about last time?

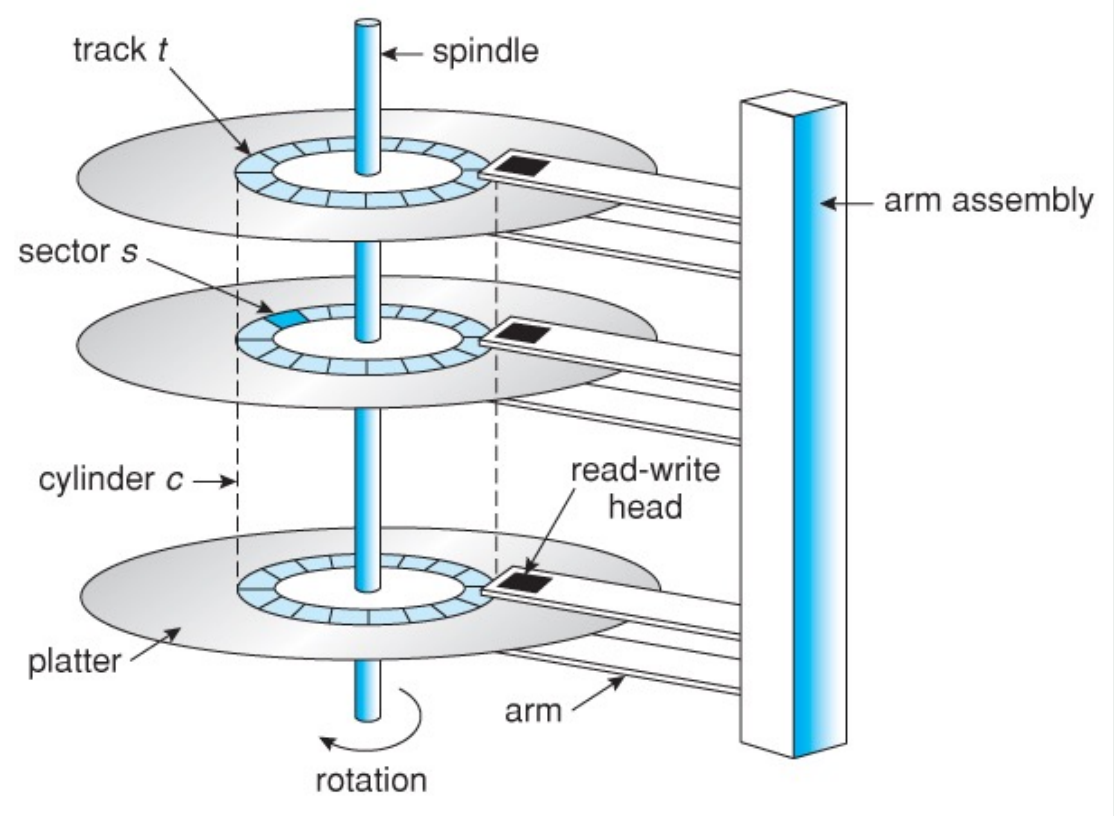
- Images
- 3-dimensional matrices
- Colors
- Blur

What will we talk about today?

- Files: Text files, binary files
- Structured Files: CSV, JSON

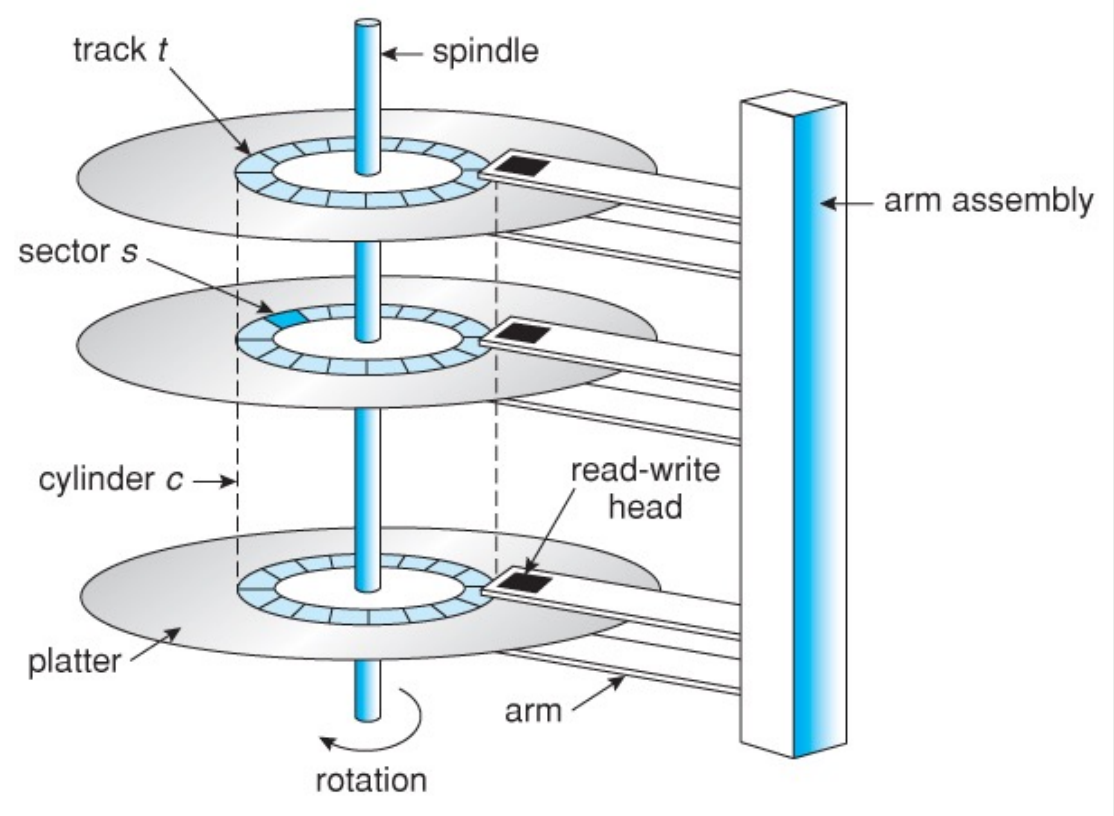
About files

- A **sequence of bytes** stored in secondary memory (HDD, SSD, etc.)
- **Persistent** storage (as long as the physical unit does not fail)
- Can have **large** size (larger than main memory)



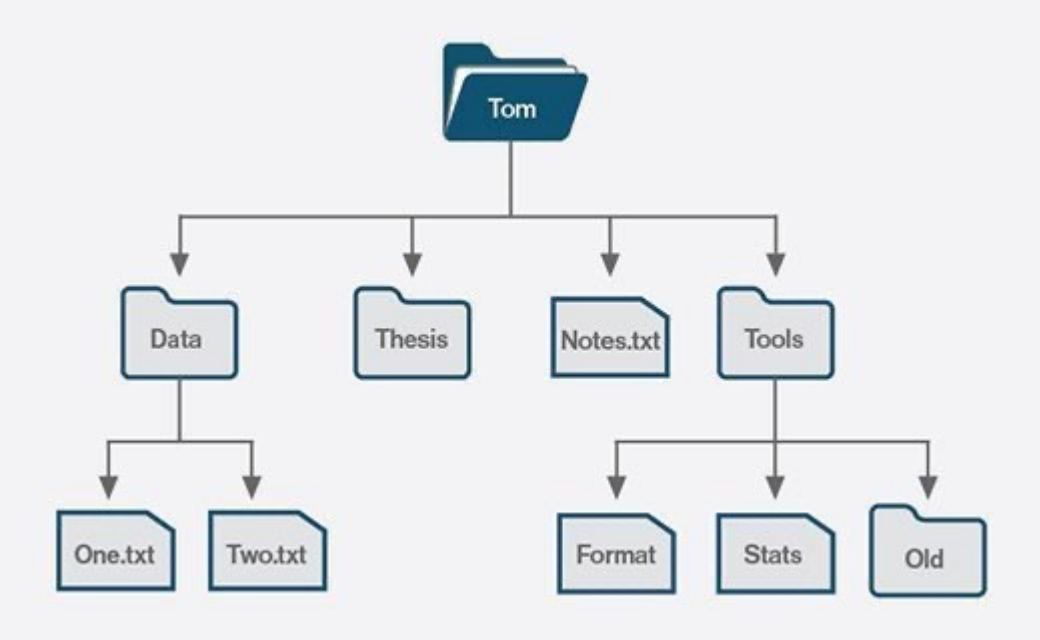
About files

- The physical unit stores data linearly, using data **blocks**.
- There is **no** explicit **structure**
- The operating system organizes the structure



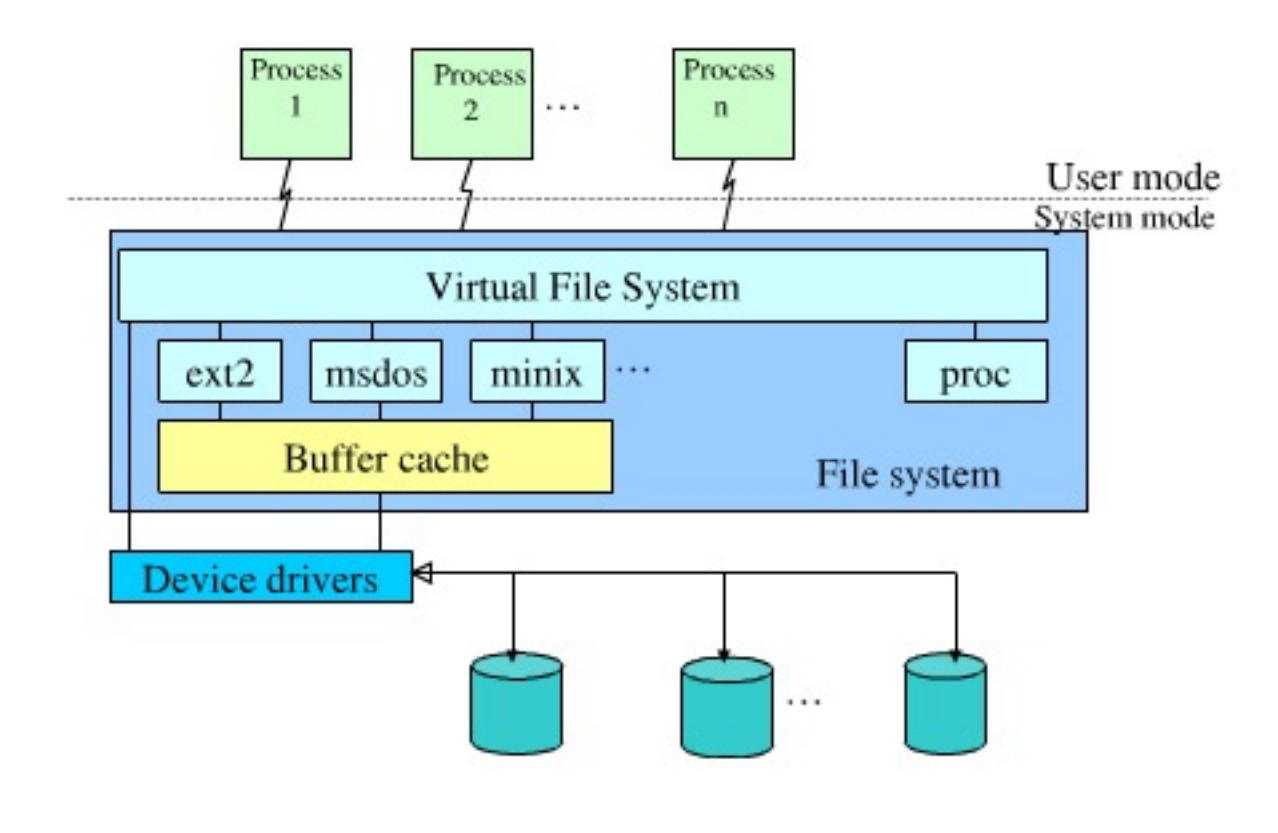
About file systems

- Use a hierarchical structure allowing for **logical grouping** of files
- Maintain a **mapping** between the logical structure and the sequence of bytes



About file systems

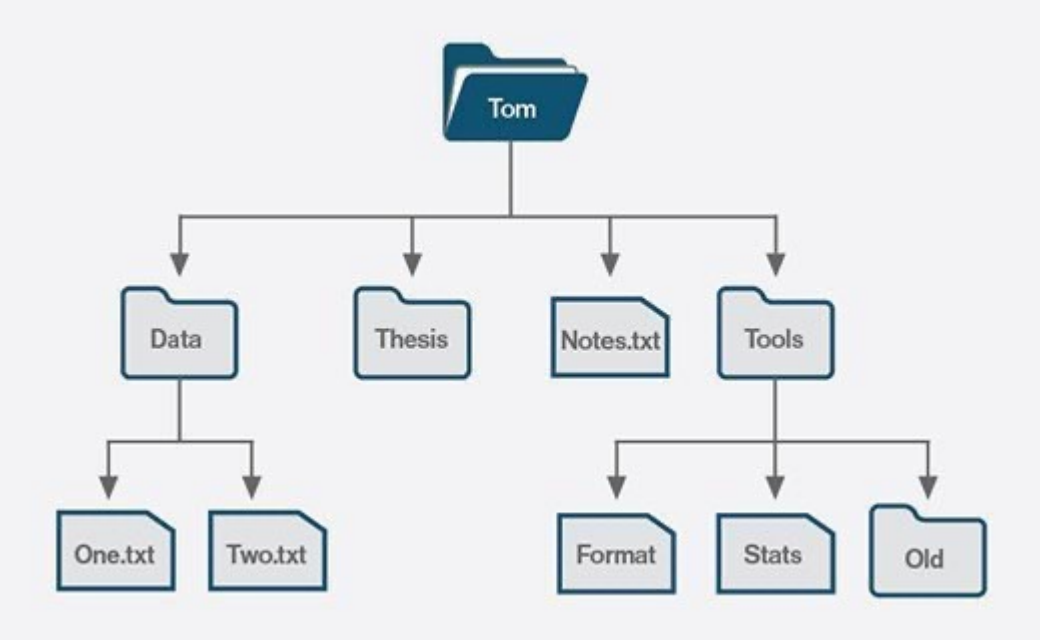
- The structure is exposed to Applications via the **Virtual File System**
- Hides implementation details
- We can focus on processing the data



You will find out more during the **Operating Systems** course

Base elements

- **Files**: atomic unit
- **Directories**: can contain other files and directories



Base operations (offered by the file system)

- **creation**:
  - creat, open (creating a file)
  - mkdir (creating a directory)
- **deletion**:
  - remove, unlink
- **renaming, moving**:
  - rename

File paths

- **Windows:**
  - C:\Documents\Newsletters\Summer2018.pdf
- **Modern operating systems:**
  - /home/adrian/Summer.pdf - linux
  - /Users/adrian/Summer.pdf - macOS

File operations

- **open**:
  - allocation of resources
  - creation of a **file handle**
  - initialization of the **file cursor**
- **read**:
  - transfer some bytes from the file to main memory
- **write**:
  - transfer some bytes from main memory to the file
- **seek:**
  - move the file **cursor** to a different position
- **close:**
  - all transfers are **synchronized**
  - allocated resources are freed up

File open mode

- **read**:
  - opens a file in read mode. The **cursor** is set at the **beginning** of the file.
  - the file must EXIST.
  - if the file does NOT exist, an error will be raised

```
F = open("example.txt", "r")
text = F.read()
F.close()
```

- **write**:
  - opens a file in write mode. The **cursor** is set at the **end** of the file.
  - If the file **exists**, it will be **overwritten**.
  - If the file does NOT exist, it will be **created**.

```
F = open("example.txt", "w")
F.write("Example text")
F.close()
```

- **append**:
  - opens a file in append mode. The **cursor** is set at the **end** of the file.
  - If the file **exists**, there is no problem
  - If the file does NOT exist, it will be **created**.

```
F = open("example.txt", "a")
F.write("Example text 2")
F.close()
```

File open mode (both read and write)

- **r+**:
  - opens a file in read mode. The **cursor** is set at the **beginning** of the file.
  - the file must EXIST.
  - if the file does NOT exist, an error will be raised
- **w+**:
  - opens a file in write mode. The **cursor** is set at the **end** of the file.
  - If the file **exists**, it will be **overwritten**.
  - If the file does NOT exist, it will be **created**.
- **a+**:
  - opens a file in append mode. The **cursor** is set at the **end** of the file.
  - If the file **exists**, there is no problem
  - If the file does NOT exist, it will be **created**.

```
F = open(“example.txt”, “r+”)
text = F.read()
F.write(text)
F.close()
```

File open mode (binary)

- **rb**:
  - opens a file in read mode. The **cursor** is set at the **beginning** of the file.
  - the file must EXIST.
  - if the file does NOT exist, an error will be raised
- **wb**:
  - opens a file in write mode. The **cursor** is set at the **end** of the file.
  - If the file **exists**, it will be **overwritten**.
  - If the file does NOT exist, it will be **created**.
- **ab**:
  - opens a file in append mode. The **cursor** is set at the **end** of the file.
  - If the file **exists**, there is no problem
  - If the file does NOT exist, it will be **created**.

Reading from text files

- **read(**max = -1**)**:
  - transfers the entire content of the file in memory.
  - problematic if the file is too large.
  - returns a **string**
  - use the argument for reading at **most max** characters
- **readline()**:
  - transfers bytes into memory until the newline symbol is encountered
  - useful if file is large
  - returns a **string**
- **readlines()**:
  - transfers the entire content of the file in memory.
  - problematic if the file is too large.
  - returns a **list of strings**, each representing one line

Each read operation moves the cursor

Anna likes apples\n

John likes walnuts\n

Mariah likes chocolate\n

```
F = open(“example.txt”, “r”)
text = F.read(5)
print(text) -> ‘Anna ‘

text2 = F.readline()
print(text2) -> ‘likes apples\n’

text3 = F.read()
print(text3) -> rest of text
F.close()
```

Writing to text files

- **write(text)**:
  - **text** must be a **string**
  - **returns** an int representing the number of bytes that can be written
  - **schedules** the transfer of bytes from main to secondary memory
  - bytes are **not** written **immediately**
  - bytes are stored into a **buffer**
  - the **OS decides** when to transfer the bytes from the buffer to disk

Moving through files

- **seek(**offset, whence = 0**)**:
  - moves the cursor to **offset**, depending on whence:
  - whence = 0: relative to the beginning of the file (positive offset)
  - whence = 1: relative to current position (positive or negative offset)
  - whence = 2: relative to end of file (negative offset)
- **tell()**:
  - returns the current position of the cursor

Write overwrites data

loves

Anna likes apples\n

John likes walnuts\n

Mariah likes chocolate\n

```
F = open(“example.txt”, “r+”)
X = F.tell()
print(X)

F.seek(5)
F.write(‘loves’)
F.close()
```

Exceptions

- What if the file does not exist?

```
F = open(“abc2.txt”, “r”)
text = F.read(5)
print(text)
```

FileNotFoundError: [Errno 2] No such file or directory: 'abc2.txt'

Exceptions

- What if the file does not exist?

```
try:
        F = open(“abc2.txt”, “r”)
        text = F.read(5)
        print(text)
        F.close()
except FileNotFountError:
        print(‘File does not exist’)
```

FileNotFoundError: [Errno 2] No such file or directory: 'abc2.txt'

Exceptions

- What if no space left?

```
try:
        F = open(“abc2.txt”, “w”)
        for i in range(10):
                F.write(“simple text”)
        F.close()
except IOError:
        print(‘Disk full’)
```

The close operation is **never** executed

IOError: [Errno 28] No space left on device:

Making sure the file is closed

- What if no space left?

```
F = None
try:
        G = open(“abc.txt”, “r”)
        F = open(“abc2.txt”, “w”)
        for i in range(10):
                F.write(“simple text”)
except FileNotFountError:
        print(‘File does not exist’)
except IOError:
        print(‘Disk full’)
finally:
        if F != None:
                F.close()
```

The **with-as** instruction

- Allows us to release the file handle and close the file auto-magically

```
try:
        with open(“abc2.txt”, “w”) as F:
                for i in range(10):
                        F.write(“simple text”)
except IOError:
        print(‘Disk full’)
```

Nested **with-as** instructions

```
try:
   with open(“input.txt”, “r”) as INF:
       with open(“abc2.txt”, “w”) as F:
              for i in INF.readlines():
                     F.write(i)

except IOError as e:
       print(‘Error’, e)
```

Reading from binary files

- **read(**max = -1**)**:
  - transfers the entire content of the file in memory.
  - problematic if the file is too large.
  - returns a **bytes** object
  - use the argument for reading at **most max** bytes

Writing to binary files

- **write(bytes)**:
  - **returns** an int representing the number of bytes that can be written
  - **schedules** the transfer of bytes from main to secondary memory
  - bytes are **not** written **immediately**
  - bytes are stored into a **buffer**
  - the **OS decides** when to transfer the bytes from the buffer to disk

bytes data type

- A sequence of values 0-255
- **to_bytes**(num_bytes, encoding)
- **from_bytes**(bytes, encoding)

```
f = open("binary.bin", "wb")

b = (12345).to_bytes(4, "big")
b2 = (9876).to_bytes(4, "big")
f.write(b)
f.write(b2)
f.close()

f = open("binary.bin", "rb")
x = f.read(4)
b = int.from_bytes(x, "big")
print(b)

x = f.read(4)
b = int.from_bytes(x, "big")
print(b)
```

CSV files

- Comma Separated Values (CSV)
- Multiple rows separated by newline (\n)
- Each row has multiple columns separated by a comma (,)

```
id,name,test1,test2
1,Adi,7,5
2,Teo,6,9
3,Ion,7,10
```

```
import csv
with open("example.csv", "r") as f: # open the file
        reader = csv.reader(f) # create a 'reader' object
        line = next(reader) # we can use next() on the object to get the next line
        print(line) # this is the header of our csv file, a list of column names
        for x in reader: print(x) # one line, a list of column values
                print(x[2]) # the grade for test2
```

JSON format

- JavaScript Object Notation (JSON)
- Popular format for transferring messages between applications written in different programming languages

```
#example.json
[{"name":"John", "age":29},
{"name":"Mark", "age":35} ]
```

```
import json
with open("example.json", "r") as f: # open the file
        data = json.load(f) # all data is loaded into memory
        print(type(data)) # is a list
        for x in data: print(x) # one object, represented as dict
                print(x['name']) # the name field of the object
```