COMP26020 Programming Languages and Paradigms -- Part 1

The C Standard Library Part 2

File I/O and Error Management

File I/O

```c
int open(const char *pathname, int flags, mode_t mode);
```

[man](https://linux.die.net/man/2/open)

- `open` creates a reference to a file: **file descriptor**
  - `pathname`: path of the file
  - `flags`:
    - Access mode: `O_RDONLY`, `O_WRONLY`, `O_RDWR`
    - Create the file if it does not exist: `O_CREAT`
    - Truncate the file size to 0 if it exists: `O_TRUNC`
    - more: https://linux.die.net/man/2/open
  - File permissions if created: `mode`

```c
// open /home/pierre/test, read-write mode, and create the file if it does not
// exists with r/w/x permissions
int file_descriptor = open("/home/pierre/test", O_RDWR | O_CREAT, S_IRWXU);
```

File I/O

```
ssize_t read(int fd, void *buf, size_t count);
```

[man](man)

- `read` attempts to read bytes from a file
    - `fd` is the file descriptor, previously created with `open`
    - `buf` is the address of the buffer that will receive the content read
    - `count` is the number of bytes to *try* to read
    - return the number of bytes *actually* read

```
int bytes_read = (file_descriptor, buffer, 10);
if(bytes_read == -1) {
    /* error */
} else if(bytes_read != 10) {
    /* technically not an error */
}
```

File I/O

```
ssize_t write(int fd, const void *buf, size_t count);
```

man

- `write` attempts to write `count` bytes from `buf` int the file corresponding to `fd`
    - Returns the number of bytes actually written

```
int bytes_written = (file_descriptor, buffer, 10);
if(bytes_written == -1) {
    /* error */
} else if(bytes_written != 10) {
    /* technically not an error */
}
```

```
int close(int fd);
```

man

- `close` terminates file I/O on a given descriptor

```c
#include <stdio.h>
#include <sys/types.h>  // needed for open
#include <sys/stat.h>   // needed for open
#include <fcntl.h>      // needed for open
#include <unistd.h>     // needed for read and write
#include <string.h>

int main(int argc, char **argv) {
    int fd1;
    char *buffer = "hello, world!";

    fd1 = open("./test", O_WRONLY | O_TRUNC | O_CREAT, S_IRUSR | S_IWUSR);
    if(fd1 == -1) {
        printf("error with open\n");
        return -1;
    }

    /* write 'hello, world!' in the file */
    if(write(fd1, buffer, strlen(buffer)) != strlen(buffer)) {
        printf("issue writing\n");
        close(fd1); return -1;
    }

    /* write it again */
    if(write(fd1, buffer, strlen(buffer)) != strlen(buffer)) {
        printf("issue writing\n");
        close(fd1); return -1;
    }

    close(fd1);
    return 0;
}
```

13-standard-library-2/file-write.c

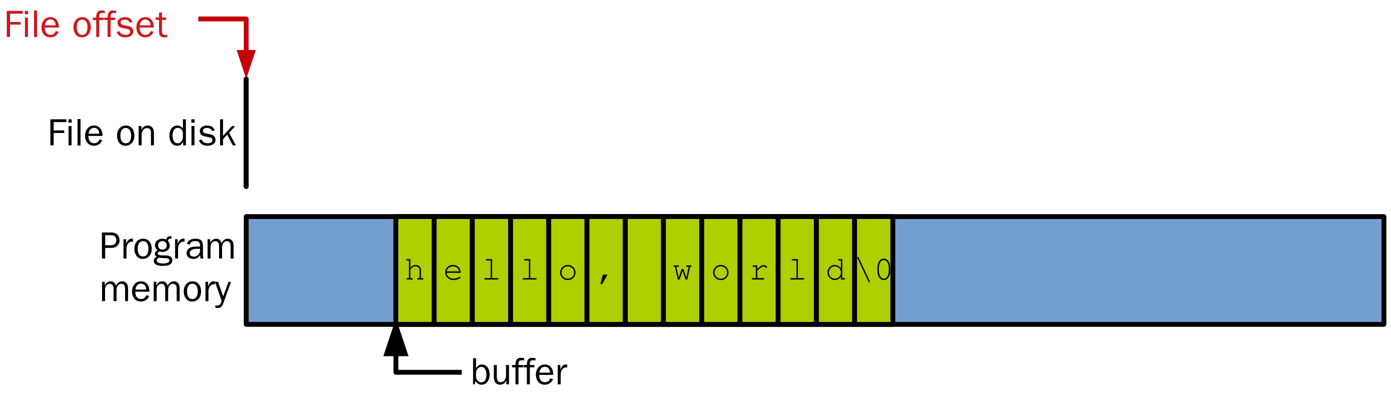
File I/O: Write

```
int fd = open( /* ... */, O_CREAT | O_TRUNC, /* ... */ );

write(fd, buffer, strlen(buffer));
write(fd, buffer, strlen(buffer));
```

File I/O: Write

```
int fd = open( /* ... */, O_CREAT | O_TRUNC, /* ... */ );

write(fd, buffer, strlen(buffer));
write(fd, buffer, strlen(buffer));
```

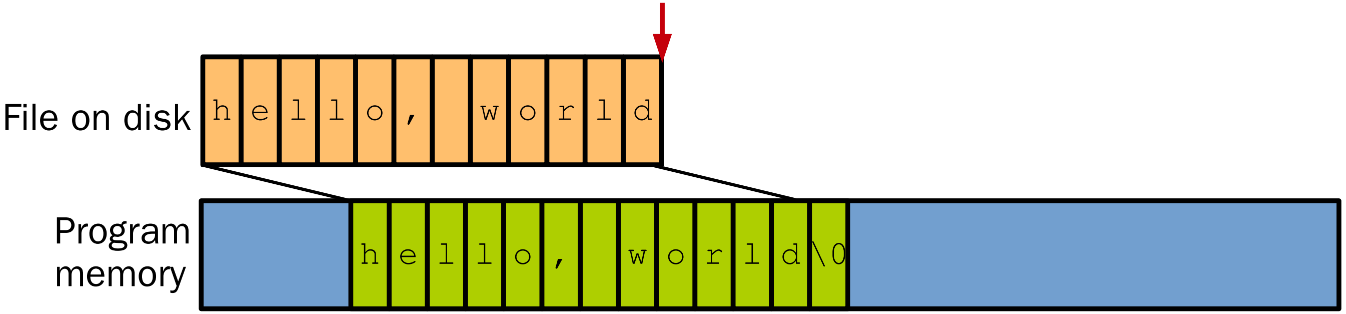
- At `open` time:



File I/O: Write

```
int fd = open( /* ... */, O_CREAT | O_TRUNC, /* ... */ );

write(fd, buffer, strlen(buffer));
write(fd, buffer, strlen(buffer));
```

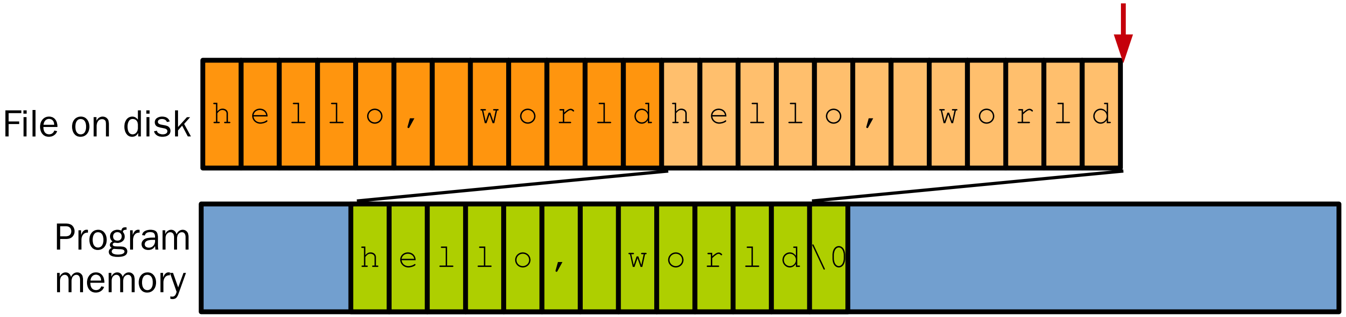
- After first `write`:



File I/O: Write

```
int fd = open( /* ... */, O_CREAT | O_TRUNC, /* ... */ );

write(fd, buffer, strlen(buffer));
write(fd, buffer, strlen(buffer));
```

- After second `write`:


File on disk
h e l l o ,   w o r l d h e l l o ,   w o r l d
Program memory
h e l l o ,   w o r l d \0

File I/O: Read

```c
// Assume ./test was previously created with the write example program
char buffer2[10];
int fd2 = open("./test", O_RDONLY, 0x0);
int bytes_read;

if(fd2 == -1) { printf("error open\n"); return -1; }

/* read 9 bytes */
if(read(fd2, buffer2, 9) != 9) {
    printf("error reading\n"); close(fd2); return -1;
}

/* fix the string and print it */
buffer2[9] = '\0';
printf("read: '%s'\n", buffer2);

/* read 9 bytes again */
bytes_read = read(fd2, buffer2, 9);
if(bytes_read != 9) {
    printf("error reading\n"); close(fd2); return -1;
}

/* fix the string and print it */
buffer2[9] = '\0';
printf("read: '%s'\n", buffer2);

close(fd2);
```

13-standard-library-2/file-read.c

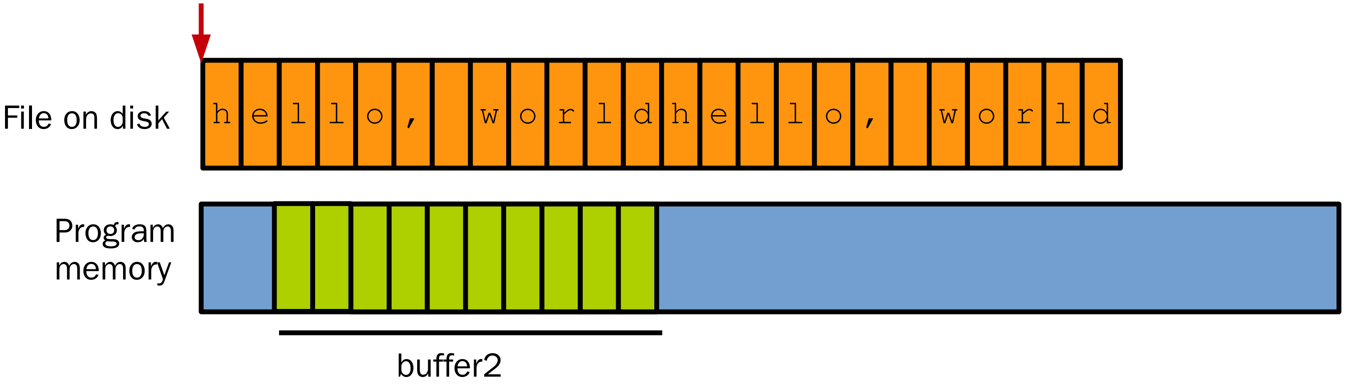
File I/O: Read

```
char buffer2[11];

int fd2 = open(/* ... */);

read(fd2, buffer2, 9);
read(fd2, buffer2, 9);
```

File I/O: Read

```
char buffer2[11];

int fd2 = open(/* ... */);

read(fd2, buffer2, 9);
read(fd2, buffer2, 9);
```

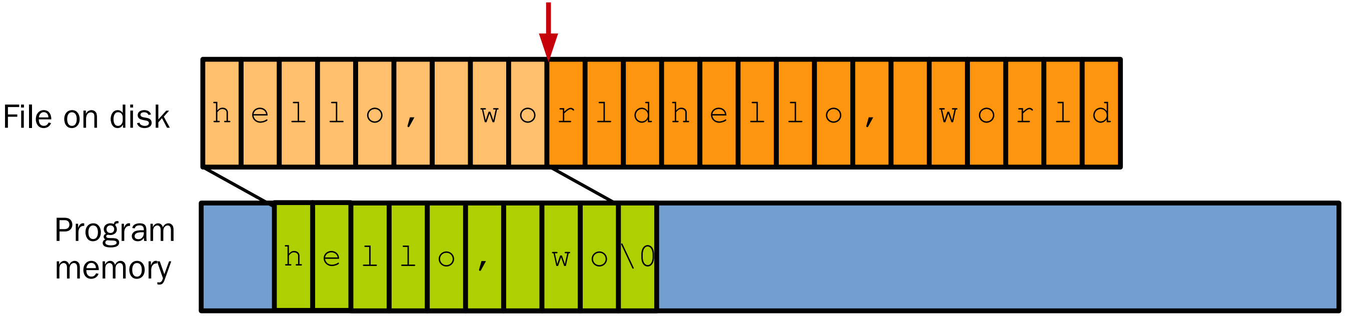
- At `open` time:



File I/O: Read

```
char buffer2[11];

int fd2 = open(/* ... */);

read(fd2, buffer2, 9);
read(fd2, buffer2, 9);
```

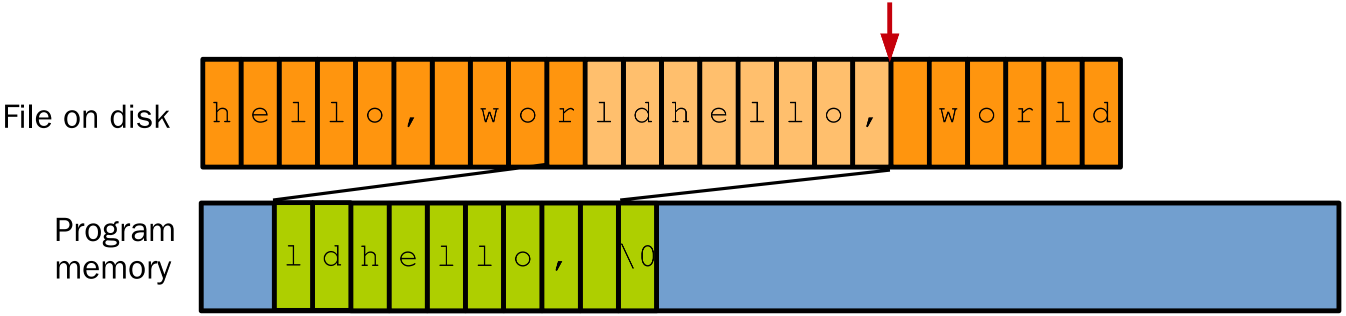
- After the first call to `read`:



File I/O: Read

```
char buffer2[11];

int fd2 = open(/* ... */);

read(fd2, buffer2, 9);
read(fd2, buffer2, 9);
```

- After the second call to `read`:



Random Numbers

Random Numbers

```c
// V1: get numbers between 0 and RAND_MAX
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    for(int i=0; i<10; i++)
        printf("%d ", rand());
    printf("\n");

    return 0;
}
```

13-standard-library-2/random-v1.c

```c
// V2: numbers between 0 and 99 with modulo
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    for(int i=0; i<10; i++)
        printf("%d ", rand()%100);
    printf("\n");

    return 0;
}
```

13-standard-library-2/random-v2.c

```c
// V3: display a different sequence each time we launch the program
int main(int argc, char **argv) {
    srand(time(NULL));  // init random seed

    for(int i=0; i<10; i++)
        printf("%d ", rand()%100);
    printf("\n");

    return 0;
}
```

13-standard-library-2/random-v3.c

Error Management

Error Management

- The variable `errno` can be used to get more information about the failure of many functions of the standard library

```c
/* ... */
#include <errno.h>  // needed for errno and perror

int main(int argc, char **argv) {
    int fd = open("a/file/that/does/not/exist", O_RDONLY, 0x0);

    /* Open always returns -1 on failure, but it can be due to many different reasons */
    if(fd == -1) {
        printf("open failed! errno is: %d\n", errno);

        /* errno is an integer code corresponding to a given reason. To format
         * it in a textual way use perror: */
        perror("open");
    }

    return 0;
}
```

13-standard-library-2/errno.c

- Errors are listed in the function man page

Summary

- File I/O
- Getting random numbers
- Error management

Feedback form: https://bit.ly/3Cv5I9W