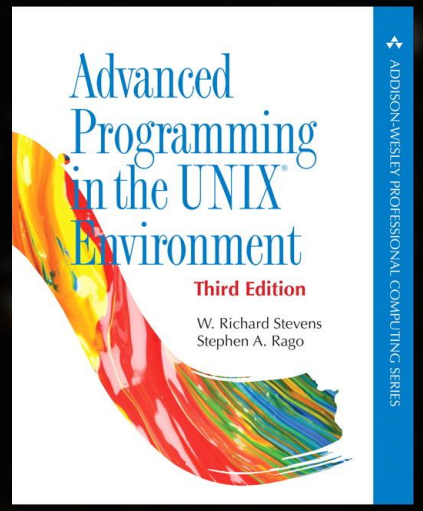
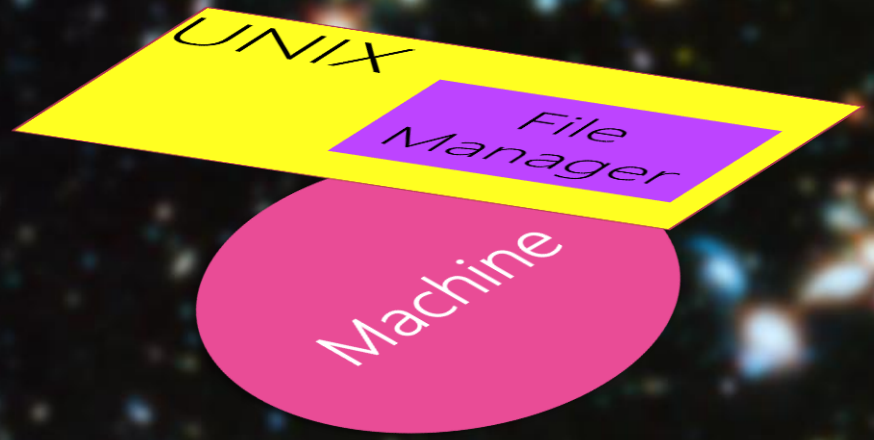
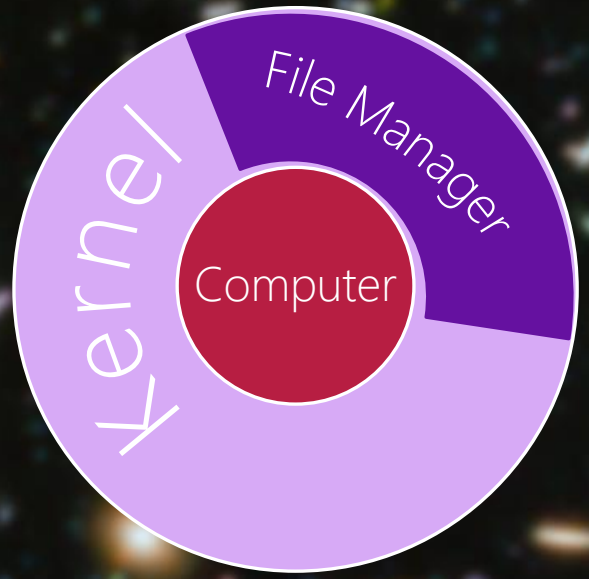

The Only Way In Or Out Is Through Phone Lines
- The Matrix (1999), Lana & Lilly Wachowski



Chapter 03: File I/O





Computer

Memory

Kernel: Device Manager

Kernel: Memory Manager

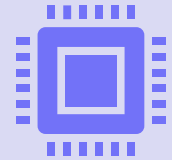
Kernel: File Manager

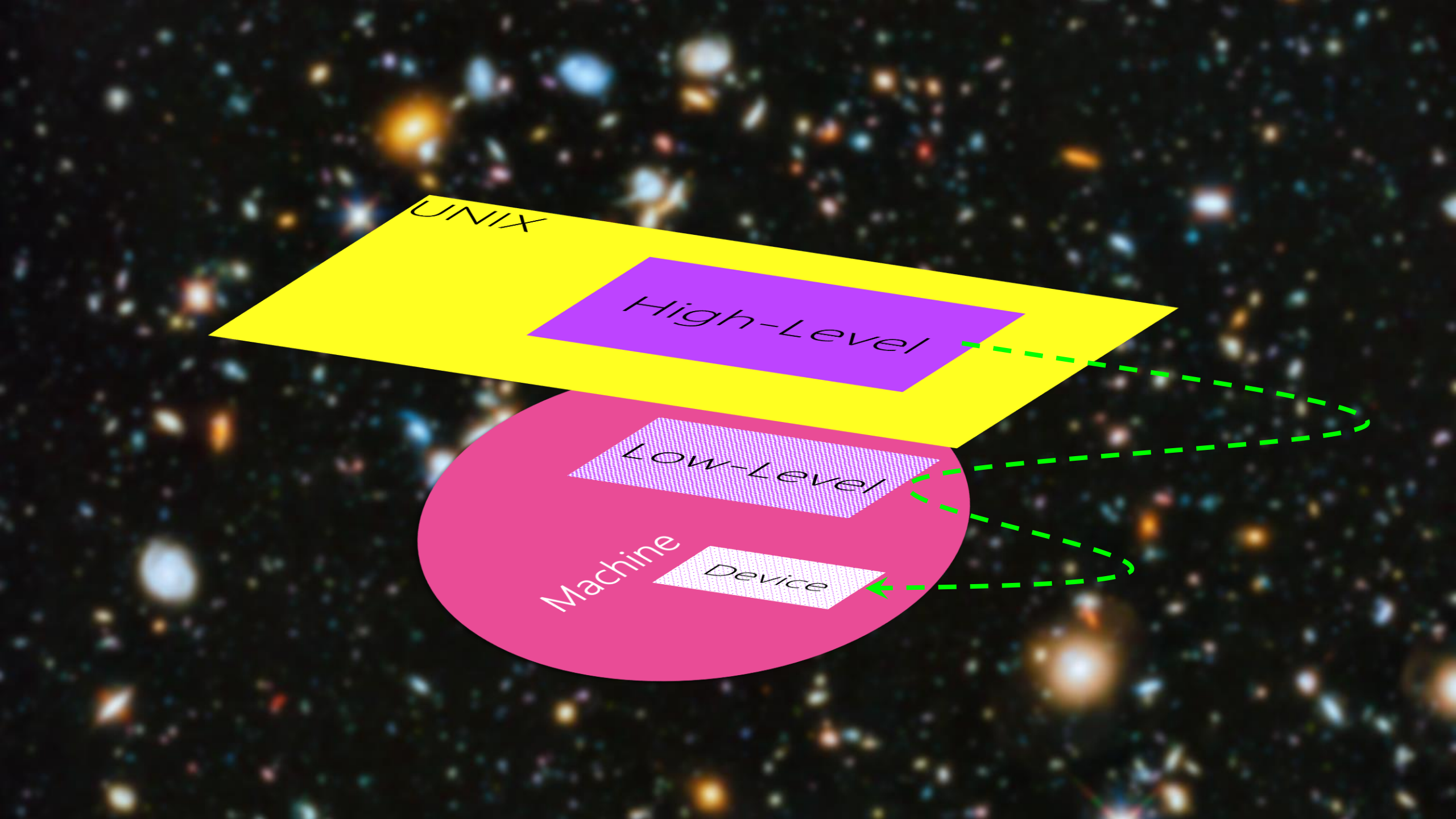
Kernel: Network Manager

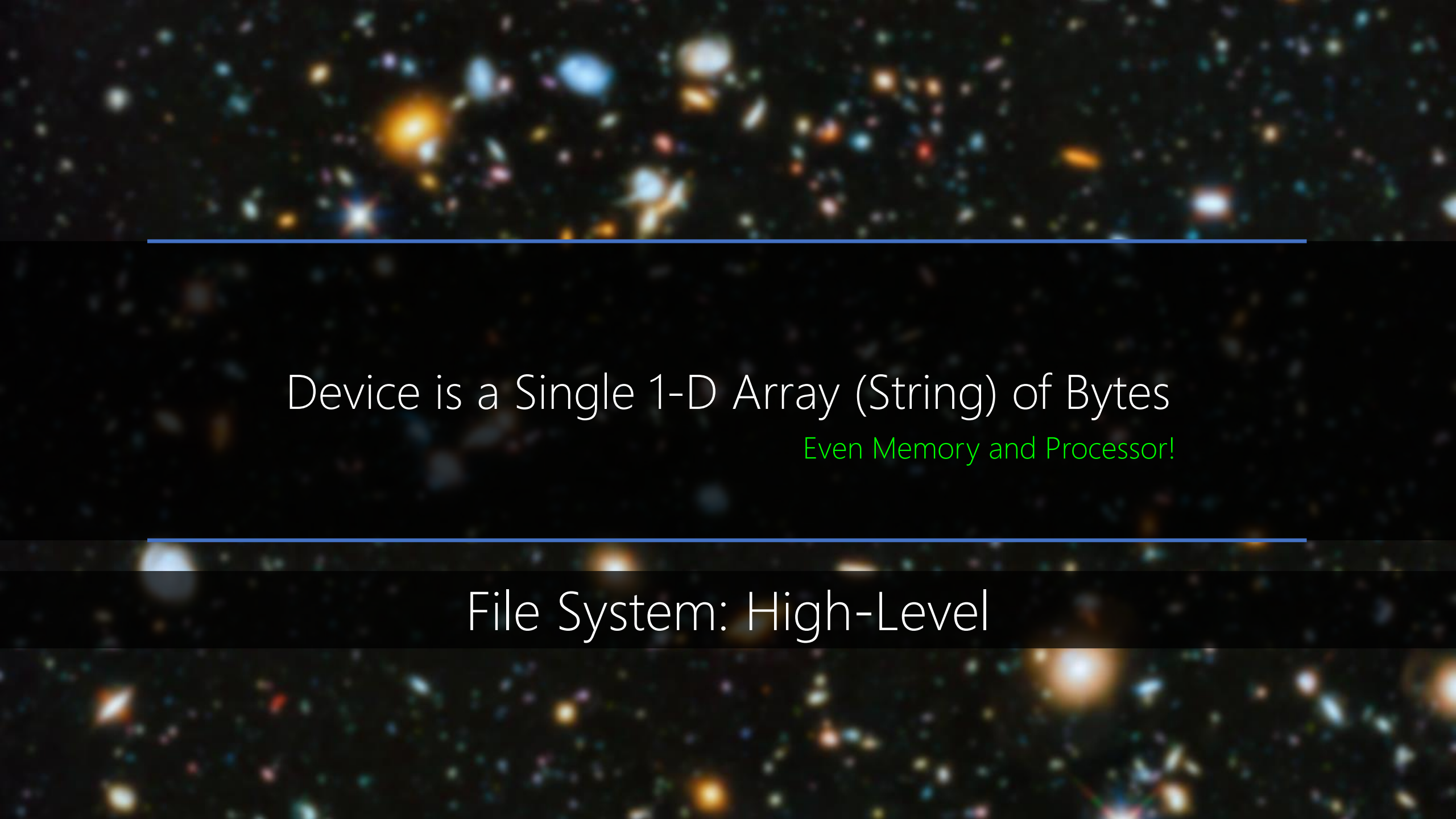
Kernel: Process Manager

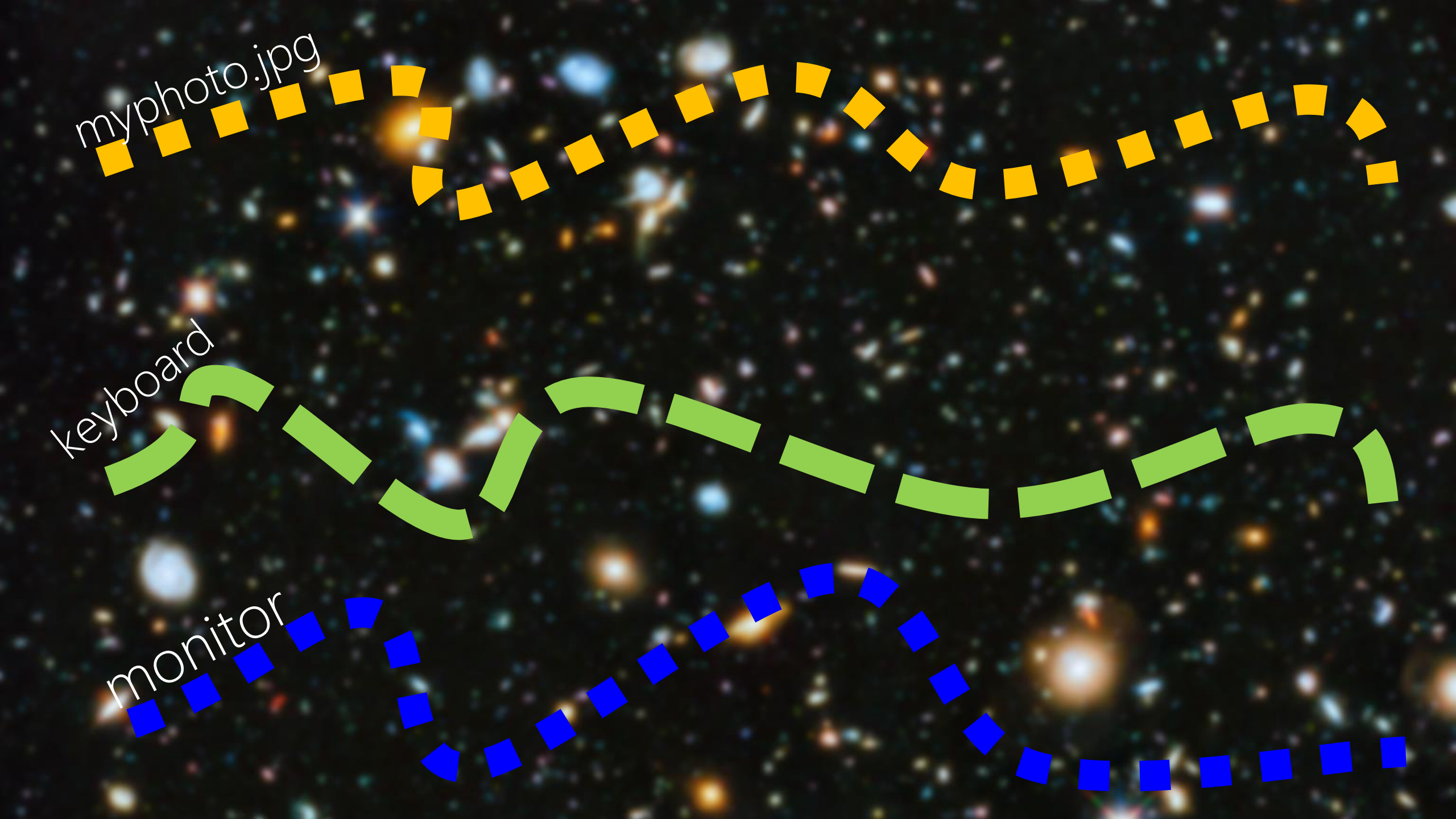
Bus

Processor




UNIX
High-Level
Low-Level
Machine
Device


Device is a Single 1-D Array (String) of Bytes
Even Memory and Processor!
File System: High-Level

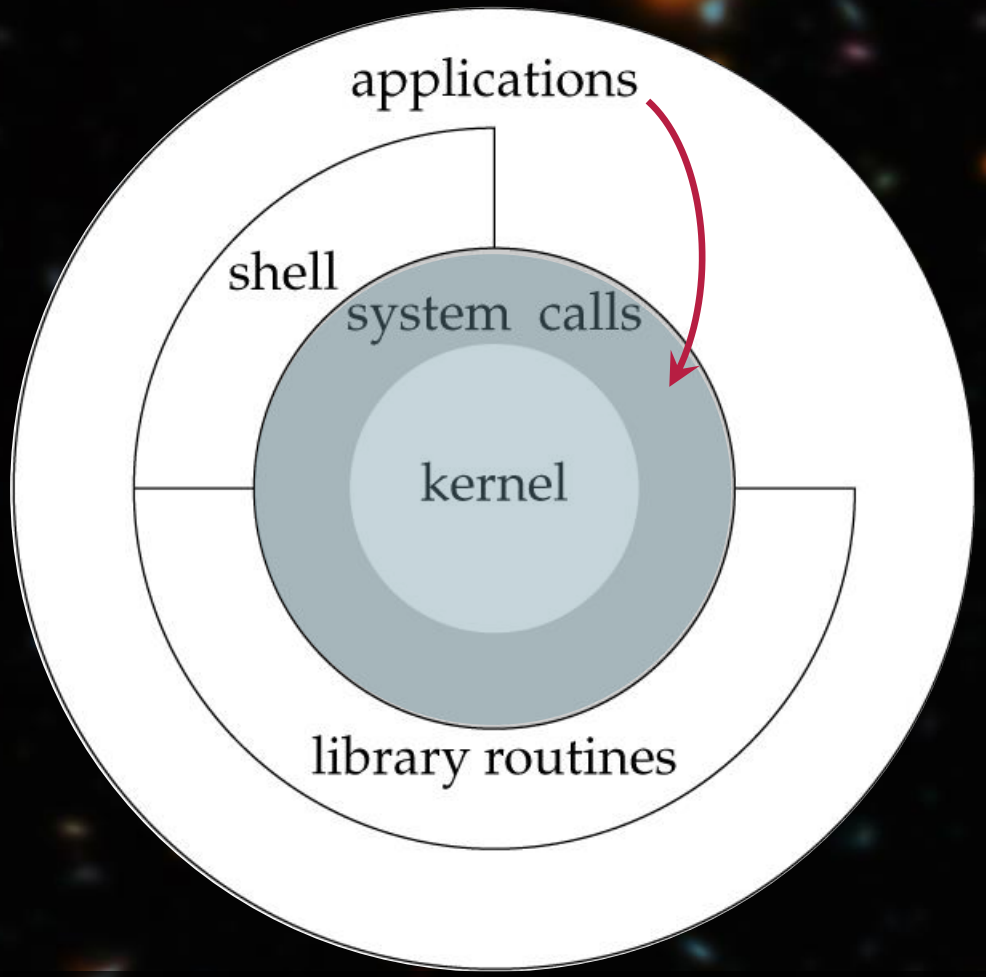

myphoto.jpg
keyboard
monitor

Operator - The Matrix (1999), Lana & Lilly Wachowski

Operation

What do you expect from a kernel about string of bytes | device | file?

File System: High-Level



Header	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Description
<aio.h>	•	•	•	•	asynchronous I/O
<cpio.h>	•	•	•	•	cpio archive values
<dirent.h>	•	•	•	•	directory entries (Section 4.22)
<dlfcn.h>	•	•	•	•	dynamic linking
<fcntl.h>	•	•	•	•	file control (Section 3.14)
<fnmatch.h>	•	•	•	•	filename-matching types
<glob.h>	•	•	•	•	pathname pattern-matching and generation
<grp.h>	•	•	•	•	group file (Section 6.4)
<iconv.h>	•	•	•	•	codeset conversion utility
<langinfo.h>	•	•	•	•	language information constants
<monetary.h>	•	•	•	•	monetary types and functions
<netdb.h>	•	•	•	•	network database operations
<nl_types.h>	•	•	•	•	message catalogs
<poll.h>	•	•	•	•	poll function (Section 14.4.2)
<pthread.h>	•	•	•	•	threads (Chapters 11 and 12)
<pwd.h>	•	•	•	•	password file (Section 6.2)
<regex.h>	•	•	•	•	regular expressions
<sched.h>	•	•	•	•	execution scheduling
<semaphore.h>	•	•	•	•	semaphores
<strings.h>	•	•	•	•	string operations
<tar.h>	•	•	•	•	tar archive values
<termios.h>	•	•	•	•	terminal I/O (Chapter 18)
<unistd.h>	•	•	•	•	symbolic constants
<wordexp.h>	•	•	•	•	word-expansion definitions
<arpa/inet.h>	•	•	•	•	Internet definitions (Chapter 16)
<net/if.h>	•	•	•	•	socket local interfaces (Chapter 16)
<netinet/in.h>	•	•	•	•	Internet address family (Section 16.3)
<netinet/tcp.h>	•	•	•	•	Transmission Control Protocol definitions
<sys/mman.h>	•	•	•	•	memory management declarations
<sys/select.h>	•	•	•	•	select function (Section 14.4.1)
<sys/socket.h>	•	•	•	•	sockets interface (Chapter 16)
<sys/stat.h>	•	•	•	•	file status (Chapter 4)
<sys/statvfs.h>	•	•	•	•	file system information
<sys/times.h>	•	•	•	•	process times (Section 8.17)
<sys/types.h>	•	•	•	•	primitive system data types (Section 2.8)
<sys/un.h>	•	•	•	•	UNIX domain socket definitions (Section 17.2)
<sys/utsname.h>	•	•	•	•	system name (Section 6.9)
<sys/wait.h>	•	•	•	•	process control (Section 8.6)

creat

POSIX

```
#include <fcntl.h>
int creat(const char *path, mode_t mode);
```

non-negative number for write-only if OK

−1 on error

File Descriptor (fs)

Number does not Matter, Connection Matters

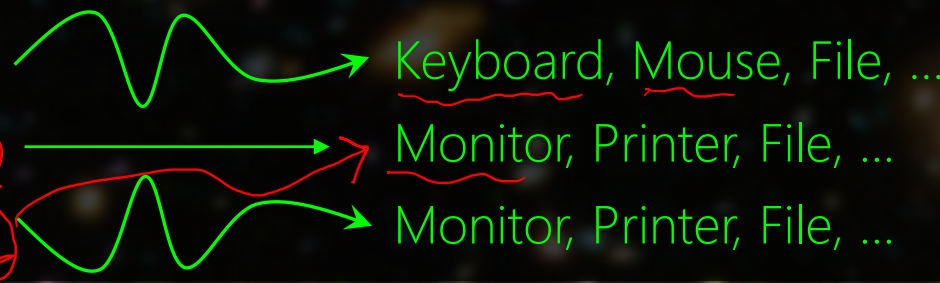
The Only Way In Or Out Is Through Phone Lines
The number does not matter, the connection is important!
Imagine a dynamic phone#, dynamic postal code, dynamic ip (DHCP)


File Descriptor (fs) != File Identifier
Because kernel reuse them for other files and devices, when available!

File Descriptor (fs)

STDIN_FILENO, STDOUT_FILENO, STDERR_FILENO

```
unistd.h
#define STDIN_FILENO  0
#define STDOUT_FILENO 1
#define STDERR_FILENO 2
```

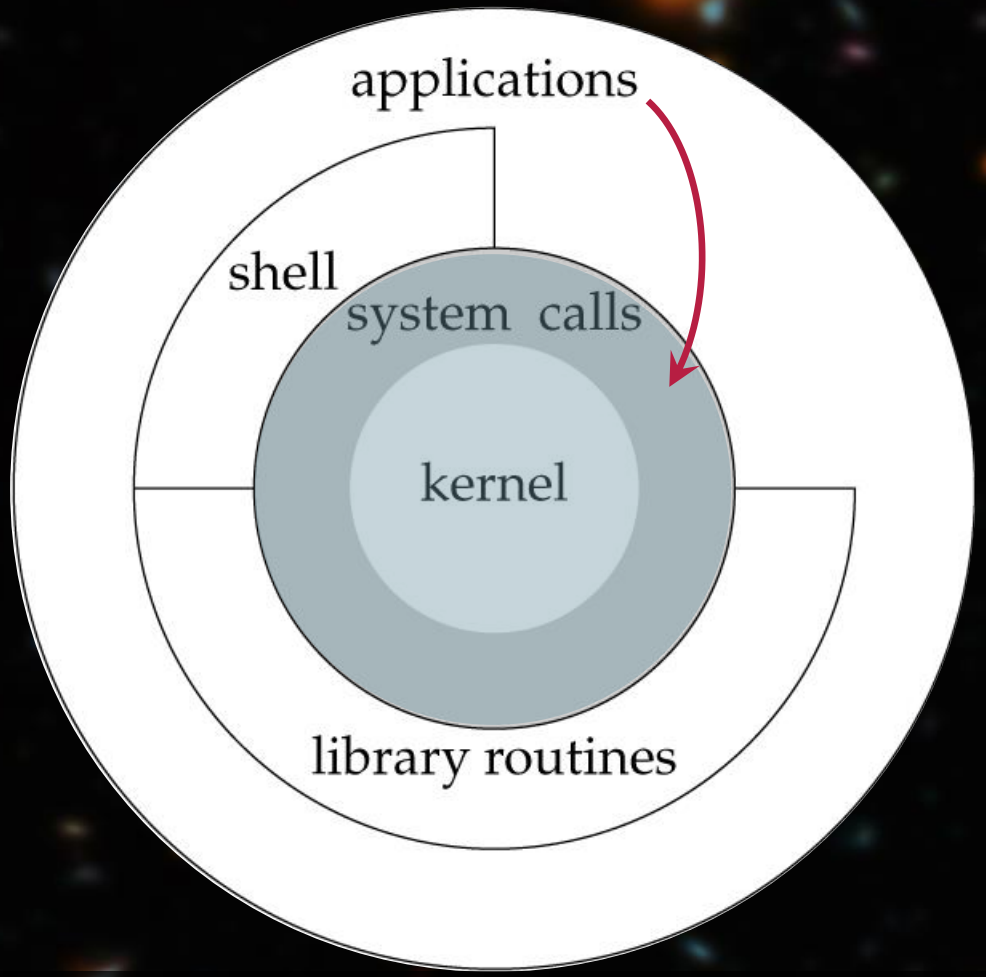


write

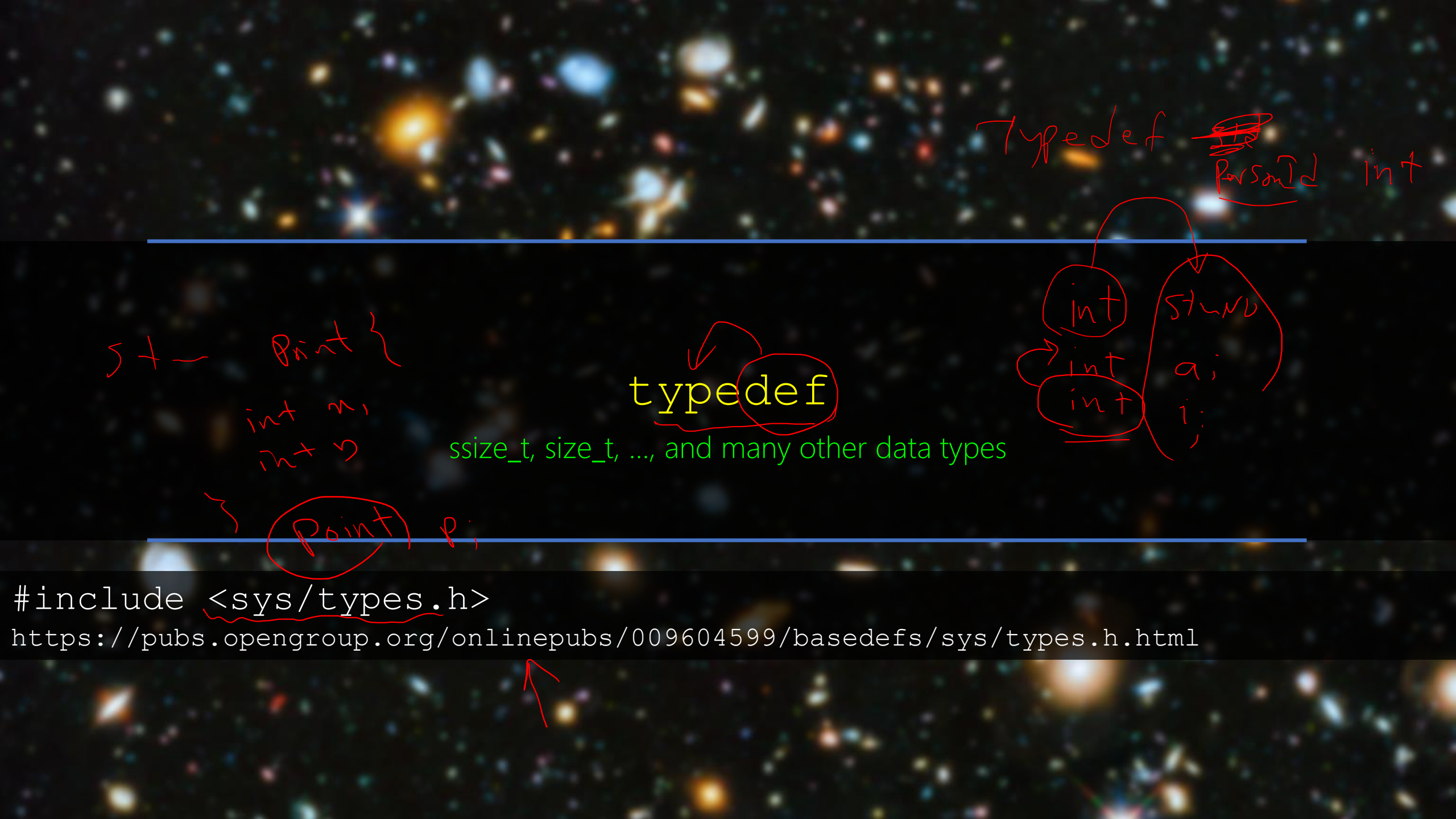
POSIX

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t nbytes);
```

number of bytes written if OK, −1 on error



Header	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Description
<aio.h>	•	•	•	•	asynchronous I/O
<cpio.h>	•	•	•	•	cpio archive values
<dirent.h>	•	•	•	•	directory entries (Section 4.22)
<dlfcn.h>	•	•	•	•	dynamic linking
<fcntl.h>	•	•	•	•	file control (Section 3.14)
<fnmatch.h>	•	•	•	•	filename-matching types
<glob.h>	•	•	•	•	pathname pattern-matching and generation
<grp.h>	•	•	•	•	group file (Section 6.4)
<iconv.h>	•	•	•	•	codeset conversion utility
<langinfo.h>	•	•	•	•	language information constants
<monetary.h>	•	•	•	•	monetary types and functions
<netdb.h>	•	•	•	•	network database operations
<nl_types.h>	•	•	•	•	message catalogs
<poll.h>	•	•	•	•	poll function (Section 14.4.2)
<pthread.h>	•	•	•	•	threads (Chapters 11 and 12)
<pwd.h>	•	•	•	•	password file (Section 6.2)
<regex.h>	•	•	•	•	regular expressions
<sched.h>	•	•	•	•	execution scheduling
<semaphore.h>	•	•	•	•	semaphores
<strings.h>	•	•	•	•	string operations
<tar.h>	•	•	•	•	tar archive values
<termios.h>	•	•	•	•	terminal I/O (Chapter 18)
<unistd.h>	•	•	•	•	symbolic constants
<wordexp.h>	•	•	•	•	word-expansion definitions
<arpa/inet.h>	•	•	•	•	Internet definitions (Chapter 16)
<net/if.h>	•	•	•	•	socket local interfaces (Chapter 16)
<netinet/in.h>	•	•	•	•	Internet address family (Section 16.3)
<netinet/tcp.h>	•	•	•	•	Transmission Control Protocol definitions
<sys/mman.h>	•	•	•	•	memory management declarations
<sys/select.h>	•	•	•	•	select function (Section 14.4.1)
<sys/socket.h>	•	•	•	•	sockets interface (Chapter 16)
<sys/stat.h>	•	•	•	•	file status (Chapter 4)
<sys/statvfs.h>	•	•	•	•	file system information
<sys/times.h>	•	•	•	•	process times (Section 8.17)
<sys/types.h>	•	•	•	•	primitive system data types (Section 2.8)
<sys/un.h>	•	•	•	•	UNIX domain socket definitions (Section 17.2)
<sys/utsname.h>	•	•	•	•	system name (Section 6.9)
<sys/wait.h>	•	•	•	•	process control (Section 8.6)




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

https://pubs.opengroup.org/onlinepubs/009604599/basedefs/sys/types.h.html

typedef

ssize_t, size_t, ..., and many other data types

```
#include <sys/types.h>
typedef size_t unsigned long
typedef ssize_t signed long
```

https://www.ibm.com/docs/en/zos/2.2.0?topic=files-systypesh

close

POSIX

```
#include <unistd.h>
int close(int fd);
```

0 if OK, −1 on error

close

fd: Releases the File Descriptor (Available for Reuse by Kernel)
No Further Access to the File (Device)

```
#include <unistd.h>
int close(int fd);
```

0 if OK, −1 on error

close

Sometimes Optional, but only Sometimes!

When a process terminates, all of its open files are closed automatically by the kernel.
That is all the File Descriptors (fs) are released.
You can take advantage of this fact and don't explicitly close open files in your programs (not recommended!)

```
hfani@alpha:~$ vi create_file_system_call.c
```

```c
#include <fcntl.h>
#include <unistd.h>

#include <sys/types.h>
#include <string.h>

#include <stdio.h>
void main(void){
        int fd;//file descriptor
        mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;//for permisison settings
        char *filename = "./my_new_file.txt";

        fd = creat(filename, mode);
        printf("The file descriptor is: %d \n", fd);

        if(fd == -1){
                printf("Error in creating file!\n");
                return;
        }

        char buf[20];
        size_t nbytes;
        ssize_t bytes_written;

        strcpy(buf, "Hello File!\n");
        nbytes = strlen(buf);

        bytes_written = write(fd, buf, nbytes);
        if(bytes_written != nbytes){
                printf("Error in writing to the file!");
        }


        int result = close(fd);
        if(result == -1){
                printf("Error in closing the file!");
        }

}
```

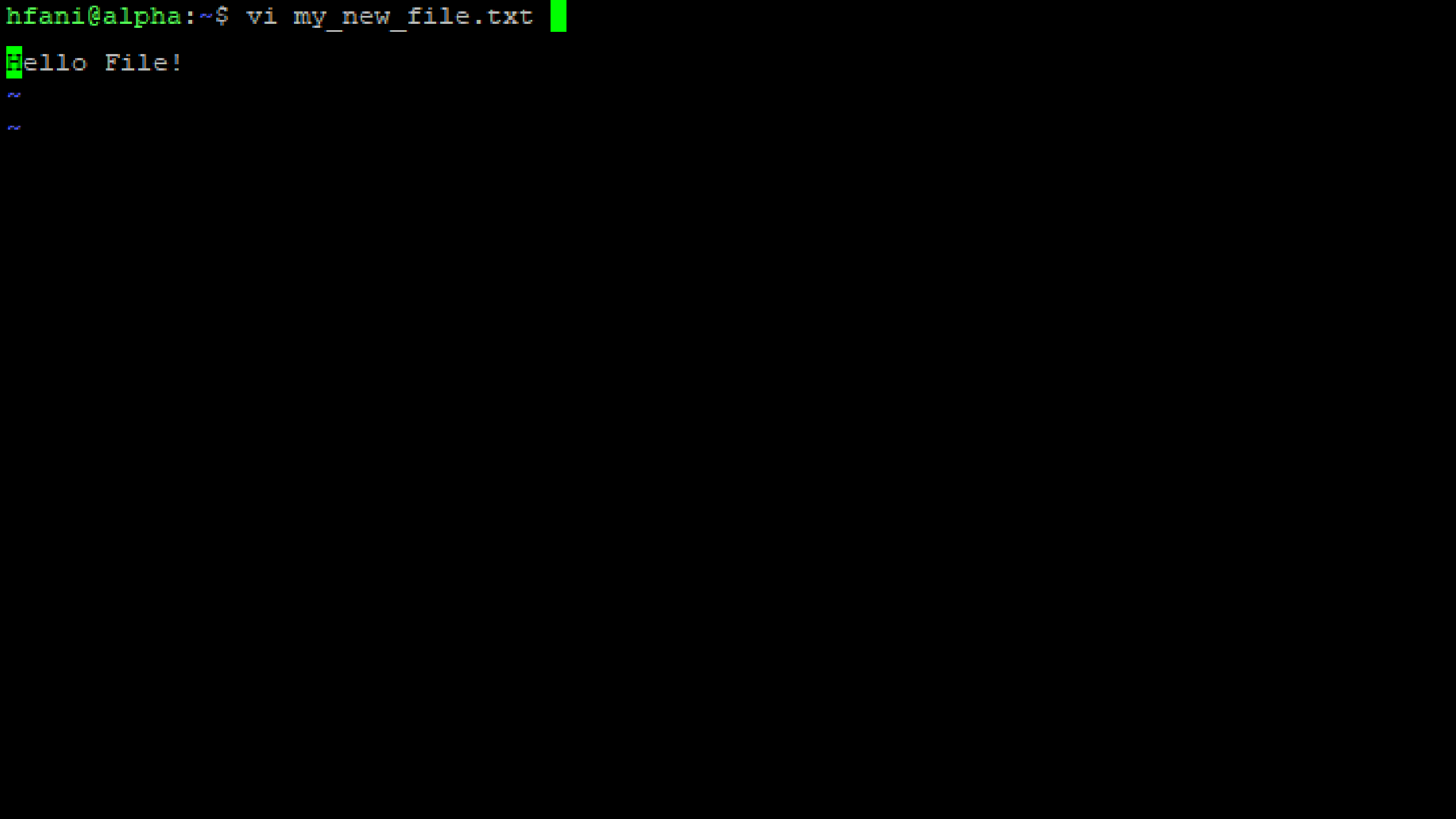
```c
#include <fcntl.h>
#include <unistd.h>

#include <sys/types.h>
#include <string.h>

#include <stdio.h>
void main(void){
        int fd;//file descriptor
        mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;//for permisison settings
        char *filename = "./my_new_file.txt";

        fd = creat(filename, mode);
        printf("The file descriptor is: %d \n", fd);

        if(fd == -1){
                printf("Error in creating file!\n");
                return;
        }

        char buf[20];
        size_t nbytes;
        ssize_t bytes_written;

        strcpy(buf, "Hello File!\n");
        nbytes = strlen(buf);

        bytes_written = write(fd, buf, nbytes);
        if(bytes_written != nbytes){
                printf("Error in writing to the file!");
        }


        int result = close(fd);
```

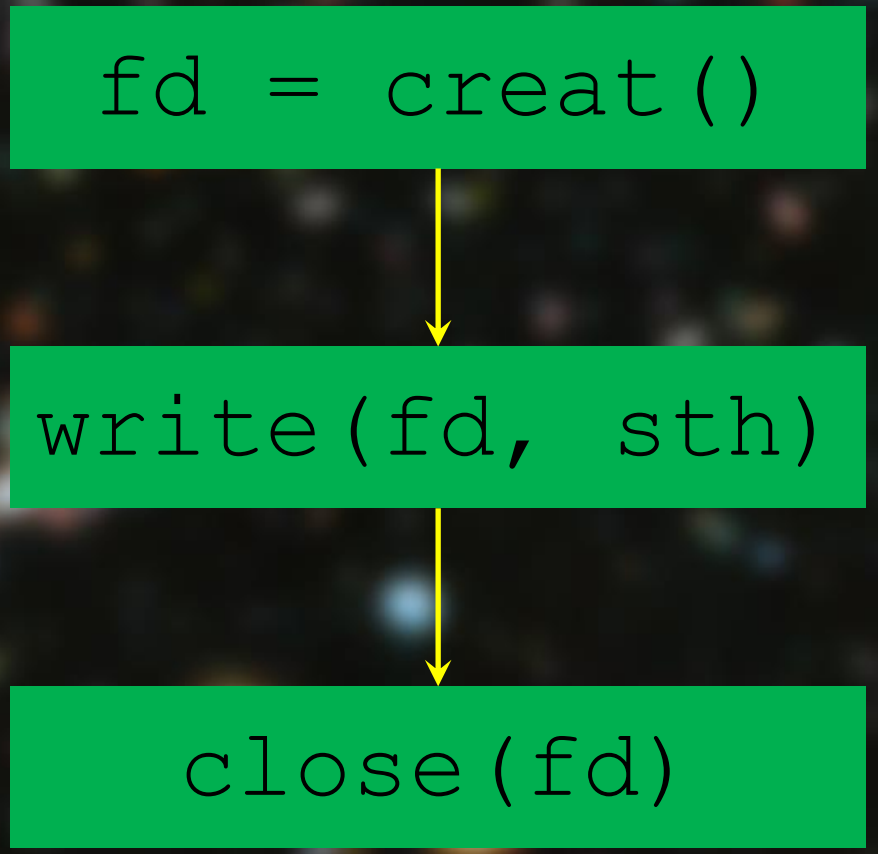
```c
#include <fcntl.h>
#include <unistd.h>

#include <sys/types.h>
#include <string.h>

#include <stdio.h>
void main(void){
        int fd;//file descriptor
        mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;//for permission settings
        char *filename = "./my_new_file.txt";

        fd = creat(filename, mode);
        printf("The file descriptor is: %d \n", fd);

        if(fd == -1){
                printf("Error in creating file!\n");
                return;
        }

        char buf[20];
        size_t nbytes;
        ssize_t bytes_written;

        strcpy(buf, "Hello File!\n");
        nbytes = strlen(buf);

        bytes_written = write(fd, buf, nbytes);
        if(bytes_written != nbytes){
                printf("Error in writing to the file!");
        }


        int result = close(fd);
```

```c
#include <fcntl.h>
#include <unistd.h>

#include <sys/types.h>
#include <string.h>

#include <stdio.h>
void main(void){
        int fd;//file descriptor
        mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;//for permission settings
        char *filename = "./my_new_file.txt";

        fd = creat(filename, mode);
        printf("The file descriptor is: %d \n", fd);

        if(fd == -1){
                printf("Error in creating file!\n");
                return;
        }

        char buf[20];
        size_t nbytes;
        ssize_t bytes_written;

        strcpy(buf, "Hello File!\n");
        nbytes = strlen(buf);

       bytes_written = write(fd, buf, nbytes);
       if(bytes_written != nbytes){
                printf("Error in writing to the file!");
       }


        int result = close(fd);
        if(result == -1){
```

```c
#include <fcntl.h>
#include <unistd.h>

#include <sys/types.h>
#include <string.h>

#include <stdio.h>
void main(void){
        int fd;//file descriptor
        mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;//for permission settings
        char *filename = "./my_new_file.txt";

        fd = creat(filename, mode);
        printf("The file descriptor is: %d \n", fd);

        if(fd == -1){
                printf("Error in creating file!\n");
                return;
        }

        char buf[20];
        size_t nbytes;
        ssize_t bytes_written;

        strcpy(buf, "Hello File!\n");
        nbytes = strlen(buf);

        bytes_written = write(fd, buf, nbytes);
        if(bytes_written != nbytes){
                printf("Error in writing to the file!");
        }


        int result = close(fd);
        if(result == -1){
                printf("Error in closing the file!");
        }
```

```
hfani@alpha:~$ cc create_file_system_call.c -o create_file_system_call
hfani@alpha:~$ ./create_file_system_call
The file descriptor is: 3
hfani@alpha:~$
```

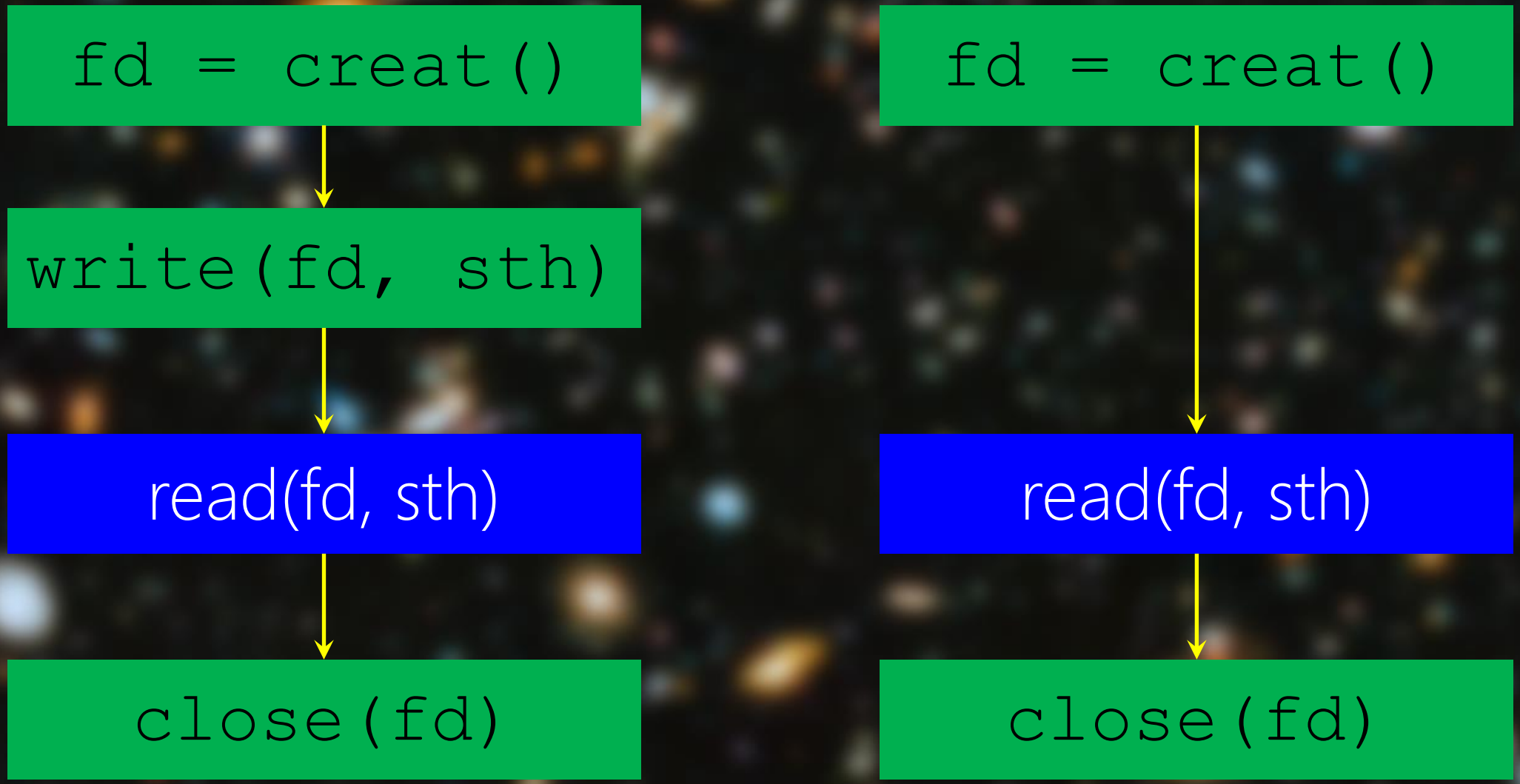

hfani@alpha:~$ vi my_new_file.txt
Hello File!
~
~

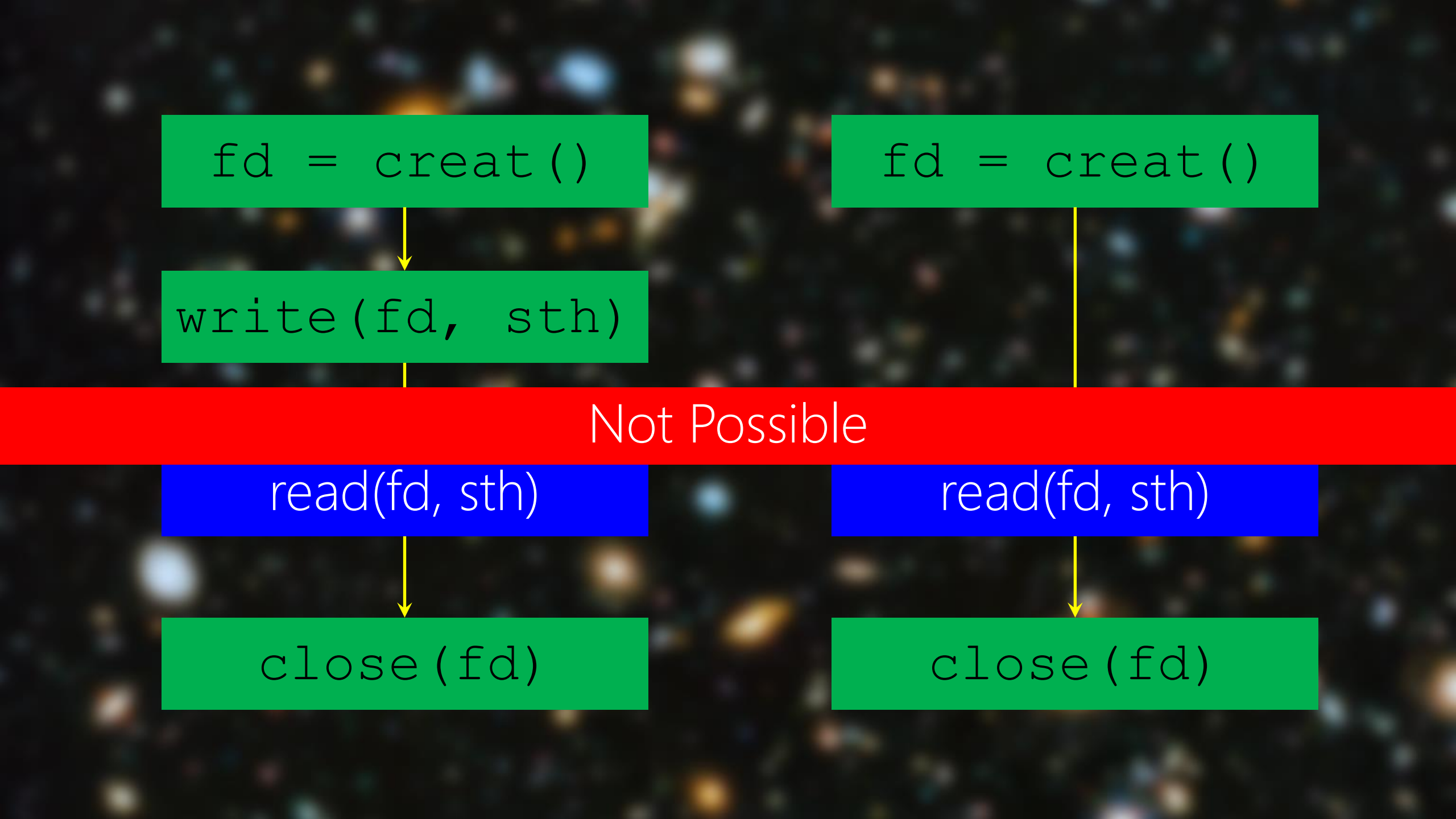

fd = creat()
write(fd, sth)
close(fd)

You Can't Handle The Truth, A Few Good Men (1992) - Aaron Sorkin

You write for later read
or
You want to read from ReadOnly File/Device

File System: High-Level


fd = creat()
write(fd, sth)
read(fd, sth)
close(fd)
fd = creat()
read(fd, sth)
close(fd)


fd = creat()
write(fd, sth)
fd = creat()
Not Possible
read(fd, sth)
read(fd, sth)
close(fd)
close(fd)

Read Not Possible w/ `creat`
What then?

File System: High-Level

open

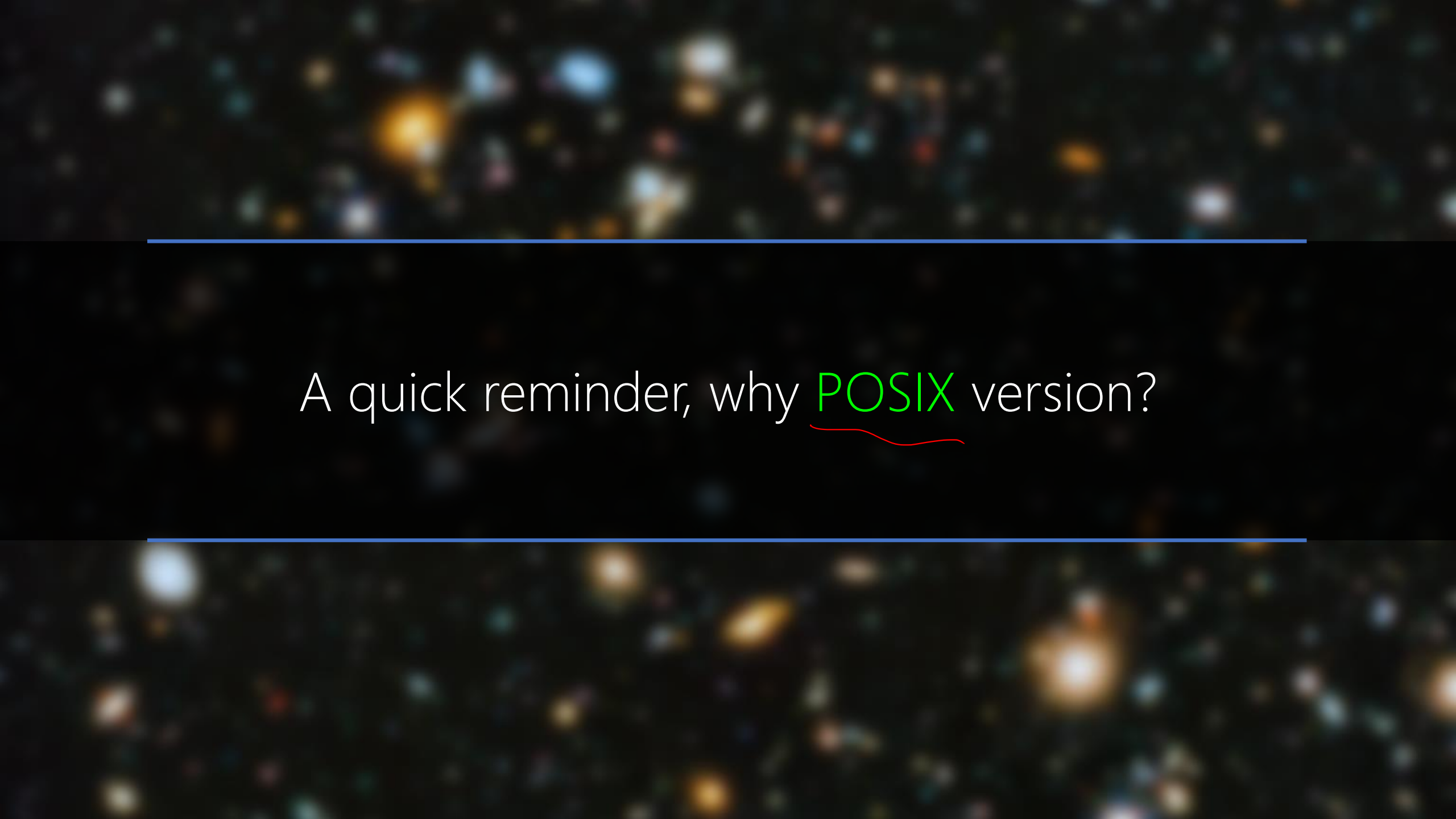
POSIX

https://pubs.opengroup.org/onlinepubs/9699919799/functions/open.html

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

−1 on error


A quick reminder, why POSIX version?

open

POSIX

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

−1 on error

open

POSIX

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

−1 on error

open

POSIX

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

−1 on error

O_RDONLY	Open for reading only (the returned fd can only read)
O_WRONLY	Open for write only (the returned fd can only write like `creat()`)
O_RDWR	Open for reading and writing (the returned fd can do both read and write)
O_EXEC	Open for execute only (the returned fd execute)
O_SEARCH	Open for search only (for directories)

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

−1 on error

O_RDONLY
O_WRONLY
O_RDWR
O_EXEC
O_SEARCH

Only One of Them

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

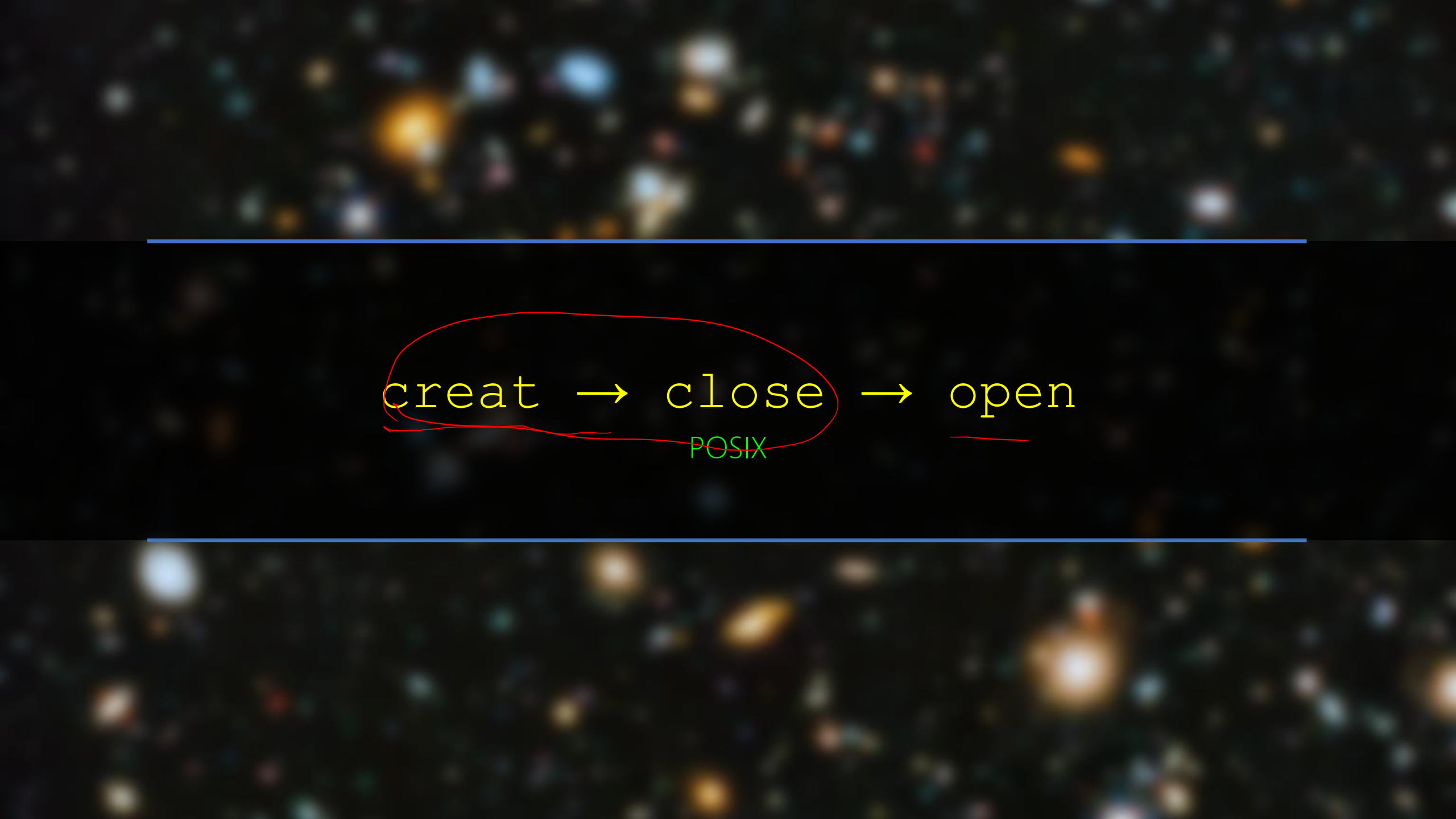
−1 on error

```
hfani@alpha:~$ vi open_rw.c
```

```c
#include <fcntl.h>
#include <stdio.h>
void main(void){
        char filename[20]="open_rw_test.txt";
        int fd = open(filename, O_RDWR);
        if(fd==-1){
                printf("error happend!\n");
        }
        else if (fd >= 0){
                printf("file successfully opened for %d and the fd is %d\n", O_RDWR, fd);
        }

}
```

```
hfani@alpha:~$ vi open_rw.c

#include <fcntl.h>
#include <stdio.h>
void main(void){
        char filename[20]="open_rw_test.txt";
        int fd = open(filename, O_RDWR);
        if(fd==-1){
                printf("error happend!\n");
        }
        else if (fd >= 0){
                printf("file successfully opened for %d and the fd is %d\n", O_RDWR, fd);
        }

}

hfani@alpha:~$ cc open_rw.c -o open_rw
hfani@alpha:~$ ./open_rw
error happend!
```

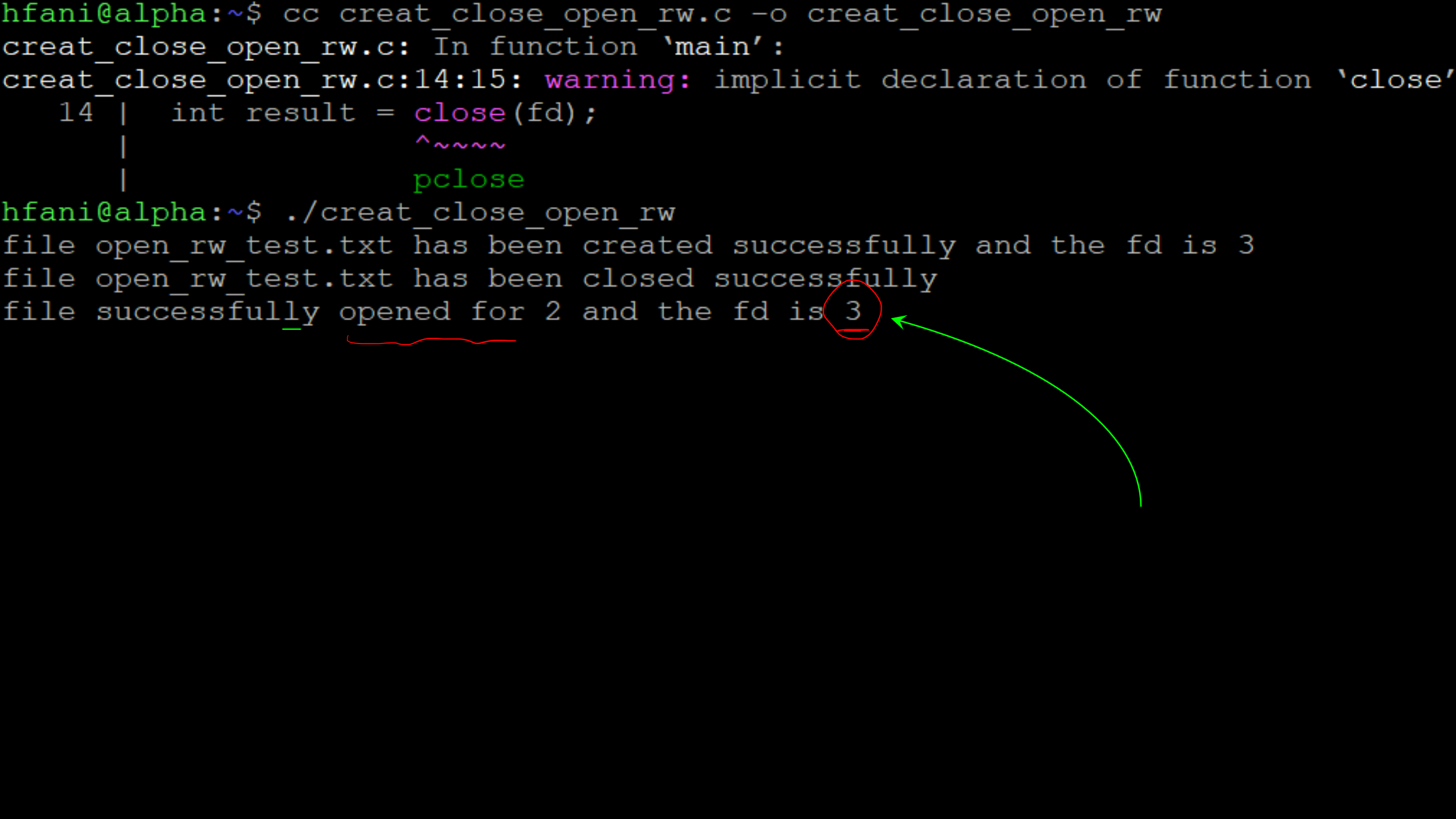
What's the problem?


creat → close → open
POSIX

```
hfani@alpha:~$ vi creat_close_open_rw.c
```

```c
#include <fcntl.h>
#include <stdio.h>
void main(void){
        char filename[20]="open_rw_test.txt";
        int fd = creat(filename, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
        if (fd == -1){
                printf("error happened in creating file %s\n", filename);
                return;
        }
        else if (fd > 0){
                printf("file %s has been created successfully and the fd is %d\n", filename, fd);
        }

        int result = close(fd);
        if (result == -1){
                printf("error happened in closing file %s\n", filename);
                return;
        }
        else if (result == 0){
                printf("file %s has been closed successfully\n", filename);
        }


        fd = open(filename, O_RDWR);
        if(fd==-1){
                printf("error happend in opening file %s!\n", filename);
        }
        else if (fd > 0){
                printf("file successfully opened for %d and the fd is %d\n", O_RDWR, fd);
        }

}
```

```
hfani@alpha:~$ cc creat_close_open_rw.c -o creat_close_open_rw
creat_close_open_rw.c: In function ‘main’:
creat_close_open_rw.c:14:15: warning: implicit declaration of function ‘close’
   14 |  int result = close(fd);
      |               ^~~~~
      |               pclose
hfani@alpha:~$ ./creat_close_open_rw
file open_rw_test.txt has been created successfully and the fd is 3
file open_rw_test.txt has been closed successfully
file successfully opened for 2 and the fd is 3
```



O_CREAT	Create the file if it does not exist (you have to specify the `mode_t`)
O_EXCL	Raise error (fd == -1) if create and the file already exists

https://stackoverflow.com/questions/48388212/what-is-the-written-out-word-for-o-excl

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

−1 on error

O_CREAT
O_EXCL

In Combination w/ the Main Five Options

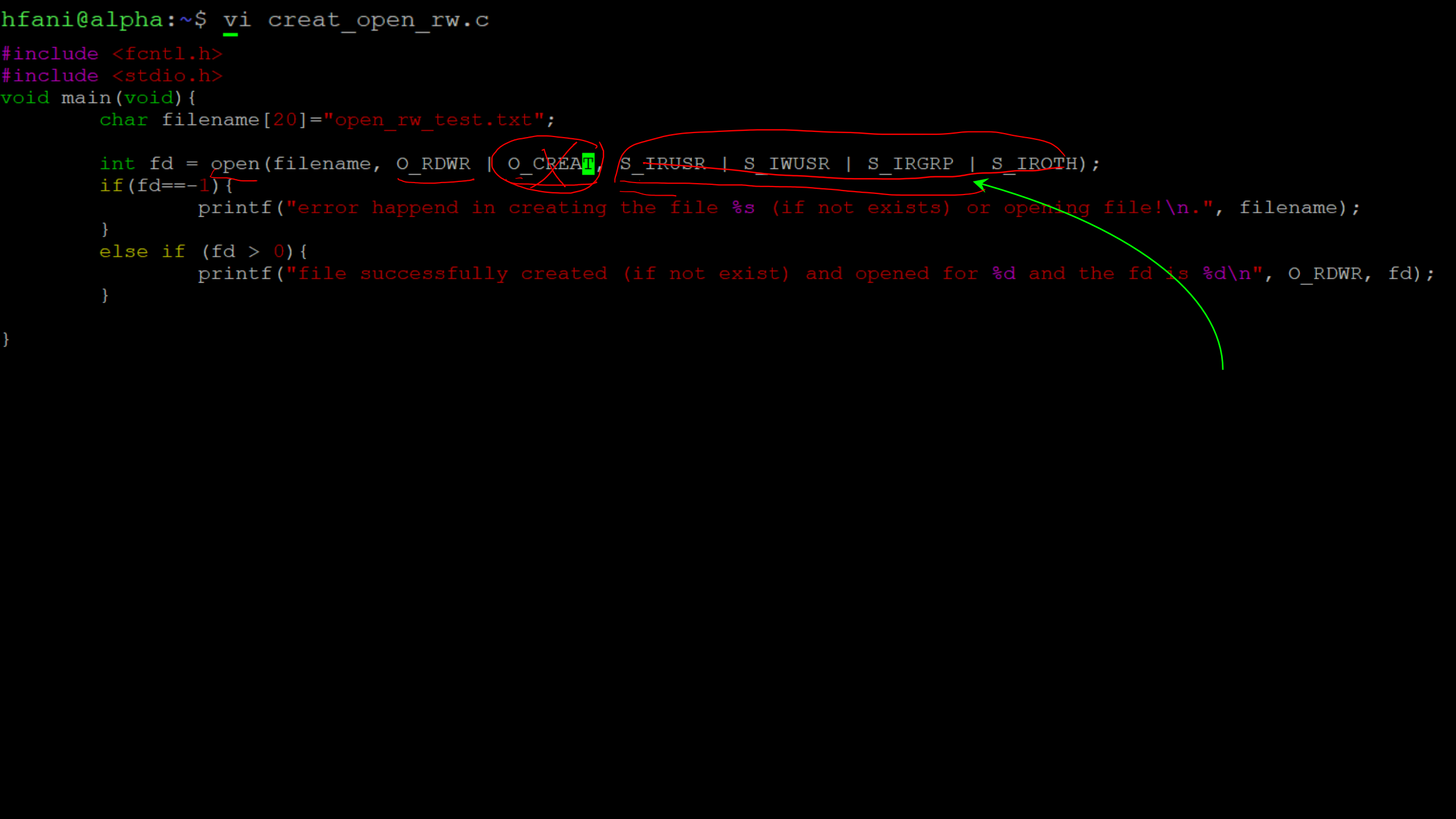
```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

−1 on error

```
hfani@alpha:~$ vi creat_open_rw.c
```

```c
#include <fcntl.h>
#include <stdio.h>
void main(void){
        char filename[20]="open_rw_test.txt";

        int fd = open(filename, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
        if(fd==-1){
                printf("error happend in creating the file %s (if not exists) or opening file!\n.", filename);
        }
        else if (fd > 0){
                printf("file successfully created (if not exist) and opened for %d and the fd is %d\n", O_RDWR, fd);
        }

}
```



```
hfani@alpha:~$ vi creat_open_rw.c
```

```c
#include <fcntl.h>
#include <stdio.h>
void main(void){
        char filename[20]="open_rw_test.txt";

        int fd = open(filename, O_RDWR | O_CREAT, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
        if(fd==-1){
                printf("error happend in creating the file %s (if not exists) or opening file!\n.", filename);
        }
        else if (fd > 0){
                printf("file successfully created (if not exist) and opened for %d and the fd is %d\n", O_RDWR, fd);
        }

}
```

```
hfani@alpha:~$ rm ./open_rw_test.txt
hfani@alpha:~$ cc creat_open_rw.c -o creat_open_rw
hfani@alpha:~$ ./creat_open_rw
file successfully created (if not exist) and opened for 2 and the fd is 3
hfani@alpha:~$ ./creat_open_rw
file successfully created (if not exist) and opened for 2 and the fd is 3
hfani@alpha:~$ ./creat_open_rw
file successfully created (if not exist) and opened for 2 and the fd is 3
hfani@alpha:~$
```

```
hfani@alpha:~$ vi creat_excl_open_rw.c
```

```c
#include <fcntl.h>
#include <stdio.h>
void main(void){
        char filename[20]="open_rw_test.txt";

        int fd = open(filename, O_RDWR | O_CREAT | O_EXCL, S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);
        if(fd==-1){
                printf("error happend in creating the file %s (may be it exists) or opening file!\n.", filename);
        }
        else if (fd > 0){
                printf("file successfully created (it didn't exist) and opened for %d and the fd is %d\n", O_RDWR, fd);
        }

}
```

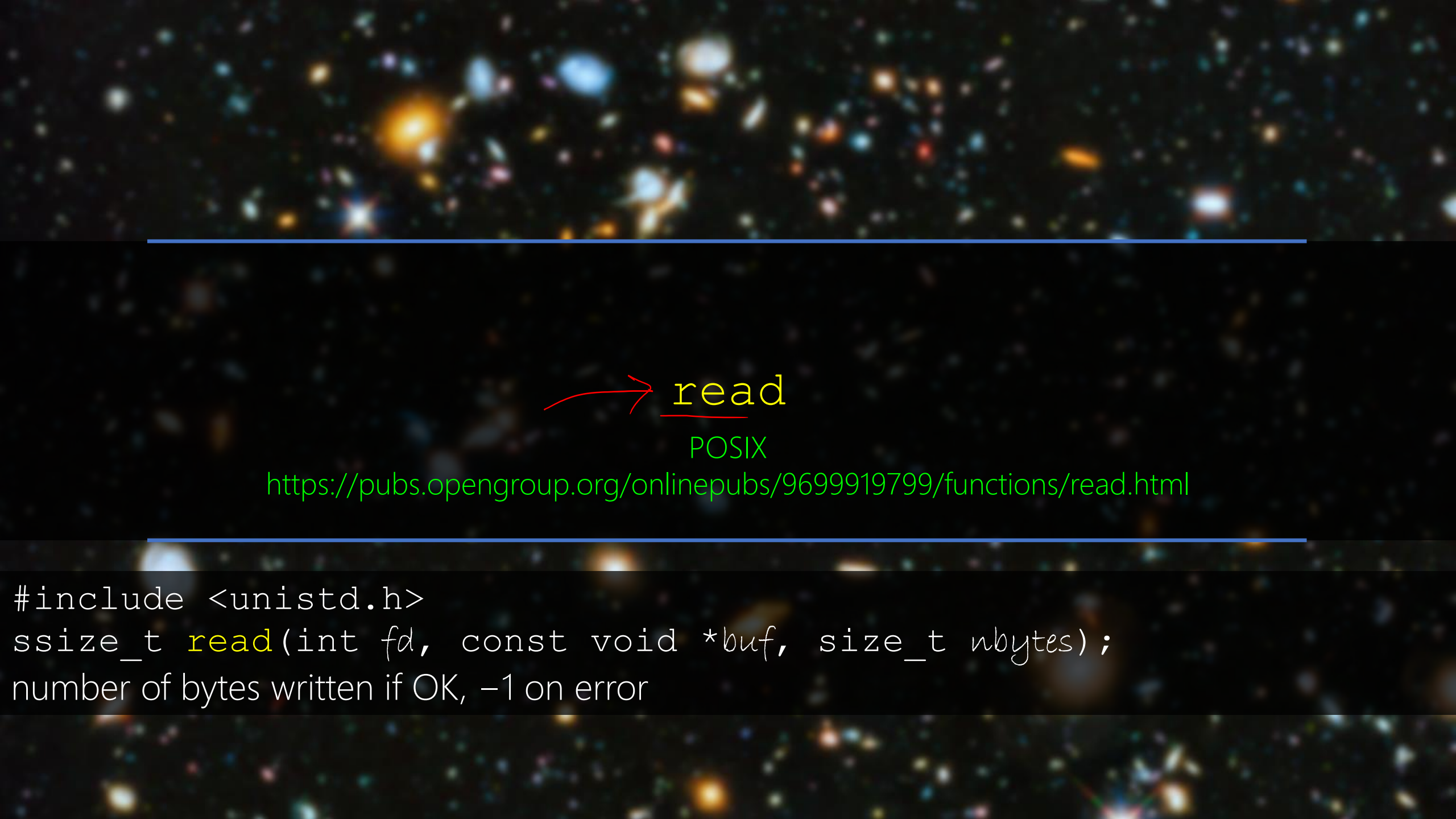
```
.hfani@alpha:~$ rm open_rw_test.txt
hfani@alpha:~$ ./creat_excl_open_rw
file successfully created (it didn't exist) and opened for 2 and the fd is 3
hfani@alpha:~$ ./creat_excl_open_rw
error happend in creating the file open_rw_test.txt (may be it exists) or opening file!
.hfani@alpha:~$ ./creat_excl_open_rw
error happend in creating the file open_rw_test.txt (may be it exists) or opening file!
```

O_APPEND Append to the end of file on each write

```
#include <fcntl.h>
int open(const char *path, int oflag, ...);
```

non-negative number (fd) if OK

−1 on error



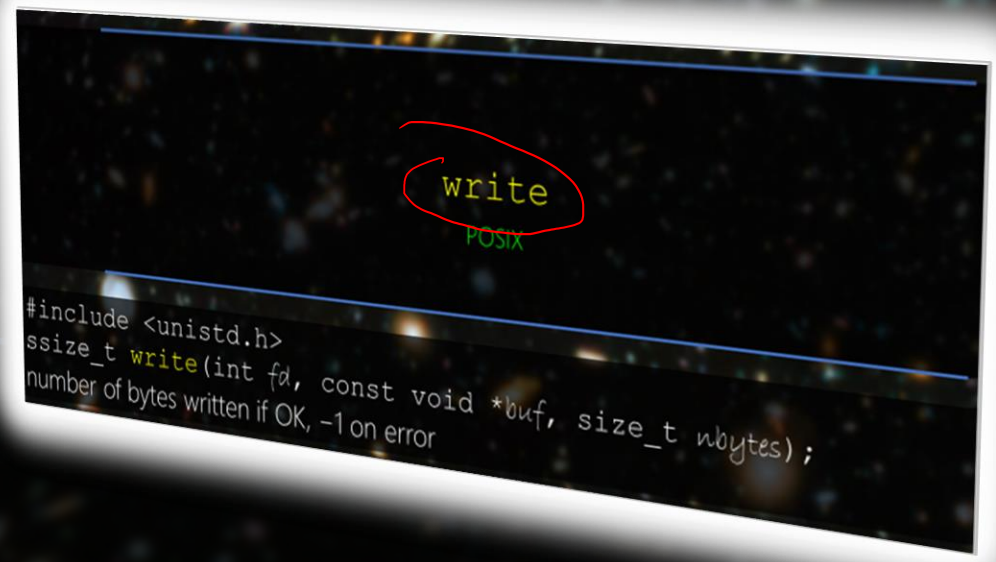
read

POSIX

https://pubs.opengroup.org/onlinepubs/9699919799/functions/read.html

```
#include <unistd.h>
ssize_t read(int fd, const void *buf, size_t nbytes);
```

number of bytes written if OK, −1 on error



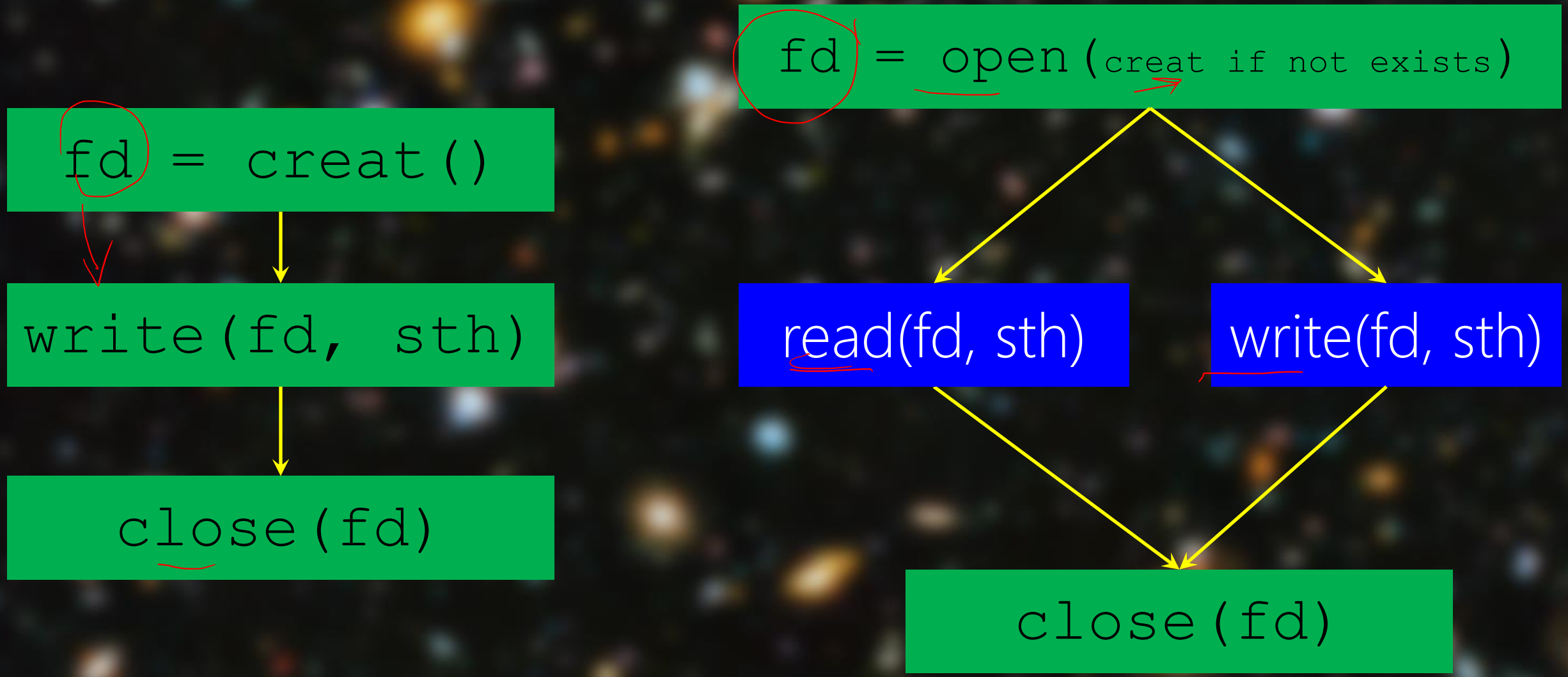
read

POSIX

https://pubs.opengroup.org/onlinepubs/9699919799/functions/read.html

```
#include <unistd.h>
ssize_t read(int fd, const void *buf, size_t nbytes);
```

number of bytes written if OK, −1 on error


fd = creat()
write(fd, sth)
close(fd)
fd = open(creat if not exists)
read(fd, sth)
write(fd, sth)
close(fd)

You Want me on that Wall, You Need me on that Wall

https://www.youtube.com/watch?v=_xd1fs5nWoI

A Few Good Men - Aaron Sorkin

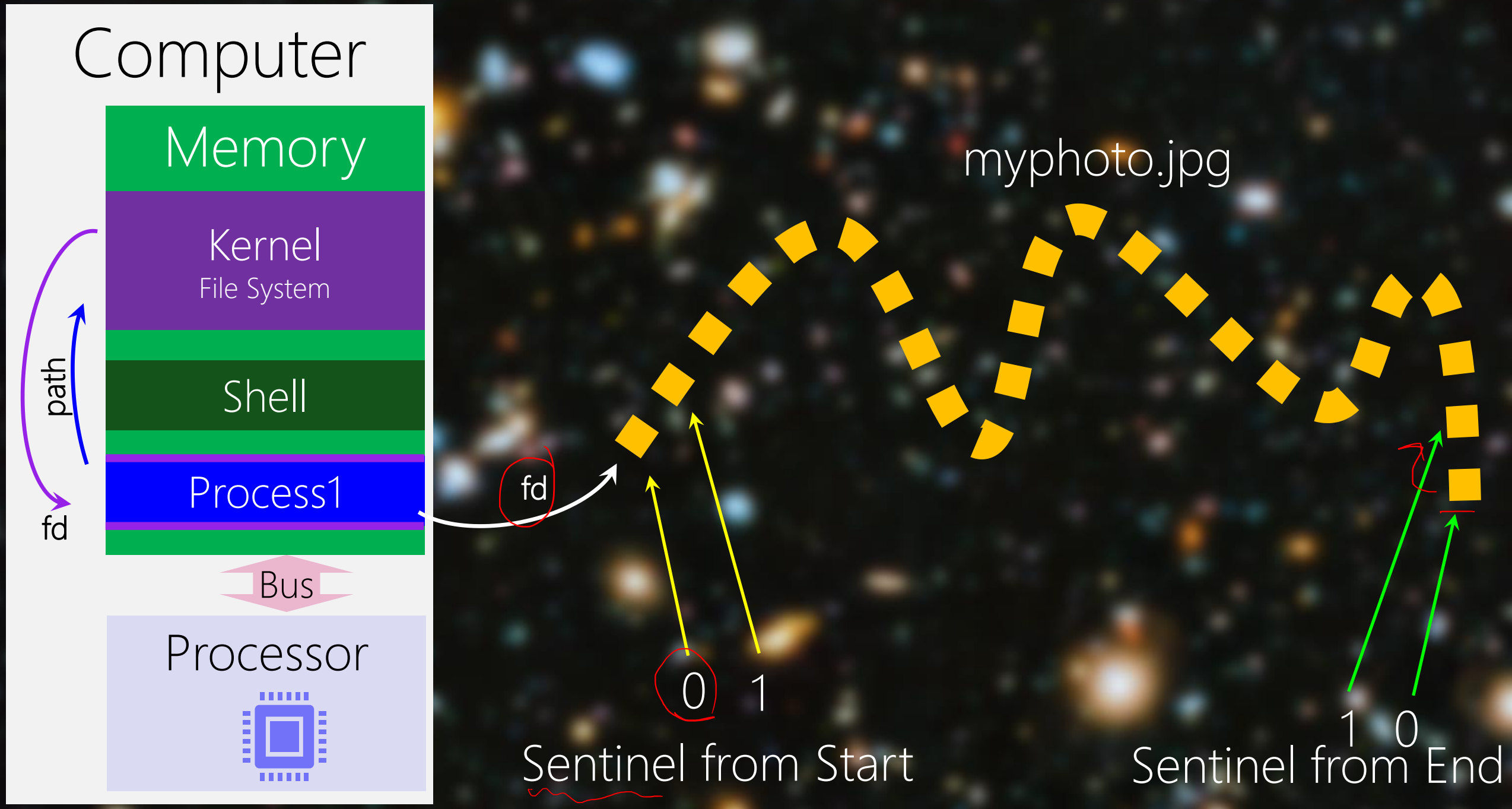
lseek

POSIX

```
#include <unistd.h>
int lseek(int fd, off_t offset, int whence);
```

file's new offset if OK (can be negative)

−1 on error


Computer
Memory
Kernel
File System
Shell
Process1
path
fd
Bus
Processor
myphoto.jpg
fd
0 1
Sentinel from Start
1 0
Sentinel from End

Every opened file has one sentinel: `current offset`

Measures the number of bytes from the beginning of the file.

Every opened file has one sentinel: `current offset`
Measures the number of bytes from the beginning of the file.

`creat()` or `open()`

Every opened file has one sentinel: `current offset`

Measures the number of bytes from the beginning of the file.

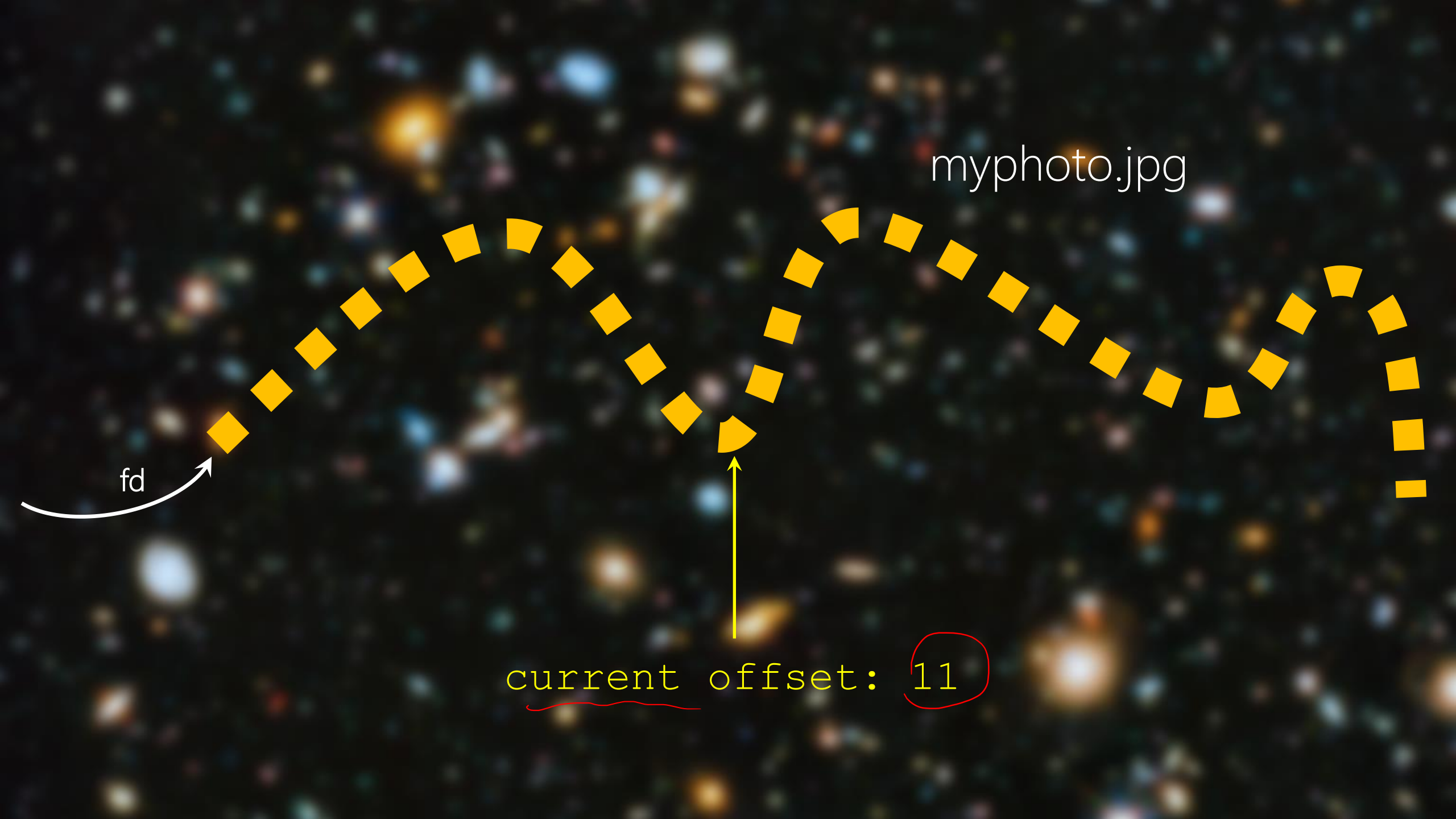
`creat()` or `open()` → 0

Every opened file has one sentinel: `current offset`

Measures the number of bytes from the beginning of the file.

`read()` or `write()` → ++ actual number of bytes read or written

`read()` or `write()` → always move forward


myphoto.jpg
fd
current offset: 11

lseek

POSIX

https://pubs.opengroup.org/onlinepubs/9699919799/functions/lseek.html

```
#include <unistd.h>
int lseek(int fd, off_t offset, int whence);
```

file's new offset if OK (can be negative)

−1 on error

lseek

POSIX

https://pubs.opengroup.org/onlinepubs/9699919799/functions/lseek.html

```
#include <unistd.h>
int lseek(int fd, off_t offset, int whence);
```

file's new offset if OK (can be negative)

−1 on error

lseek

How many bytes move the current offset

```
#include <unistd.h>
int lseek(int fd, off_t offset, int whence);
```

file's new offset if OK (can be negative)

−1 on error

```
#include <sys/types.h>
typedef off_t signed long
```

lseek

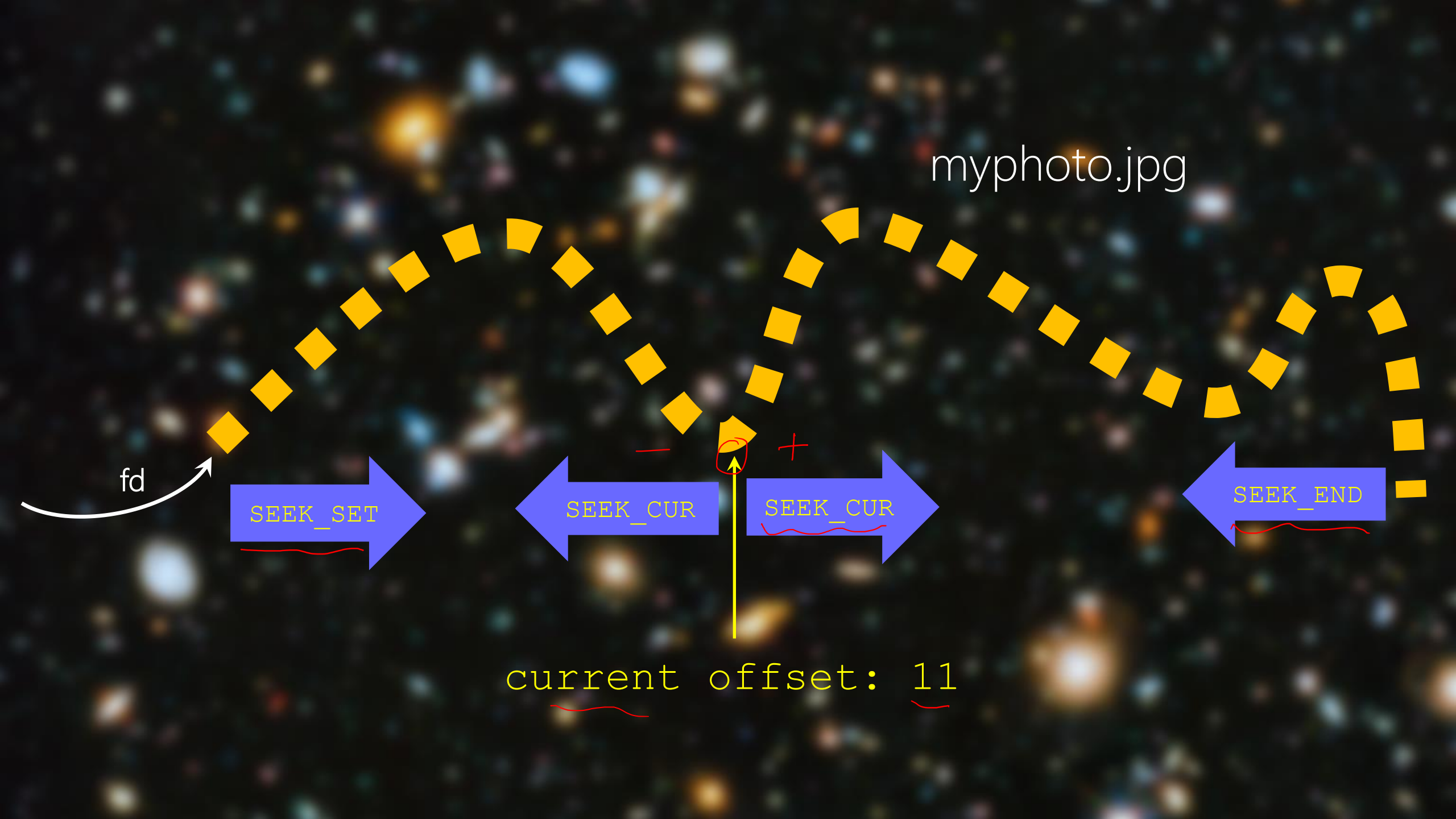
How many bytes move the `current offset` from what place or origin?

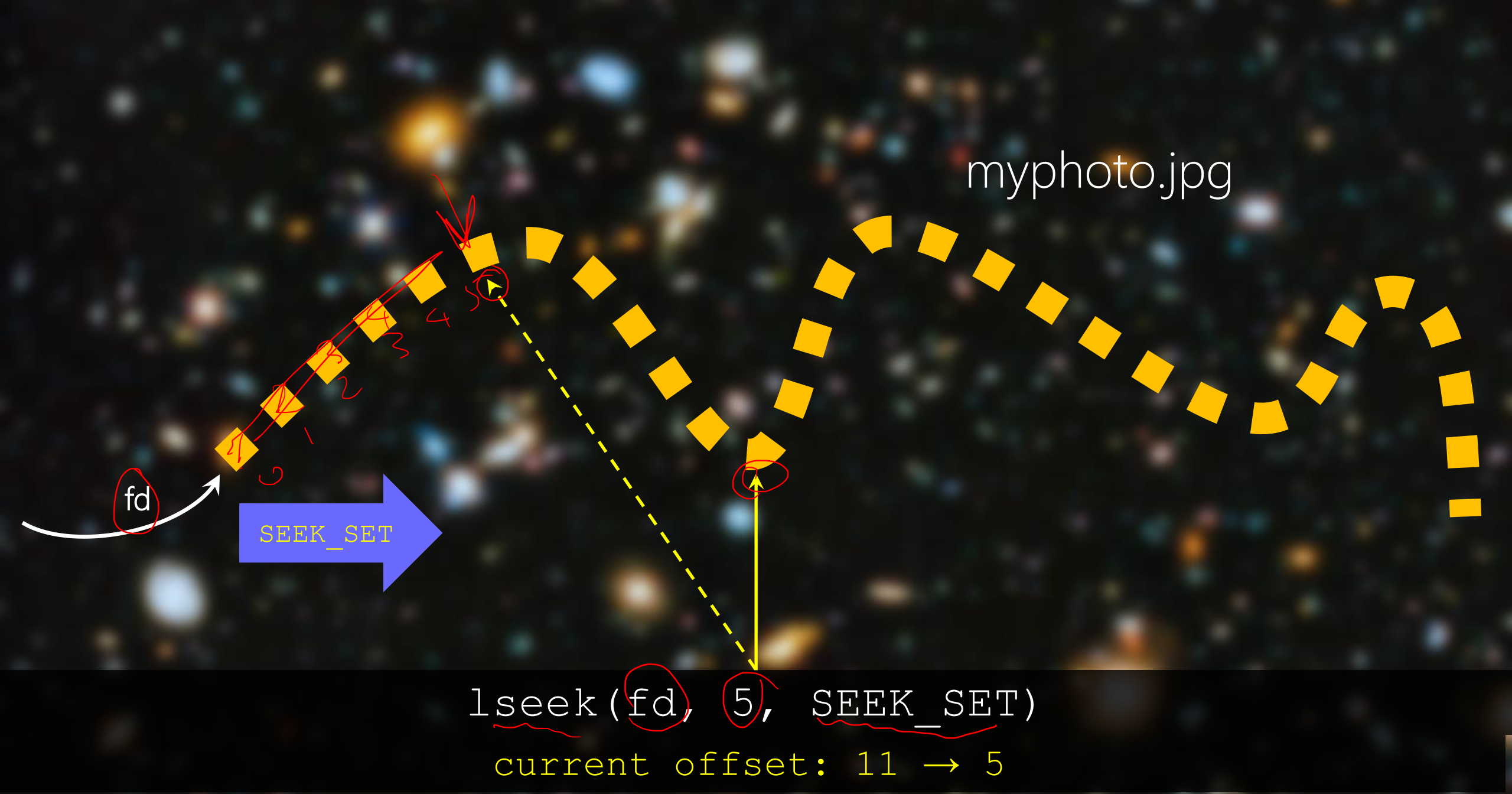
`SEEK_SET`, `SEEK_CUR`, `SEEK_END`

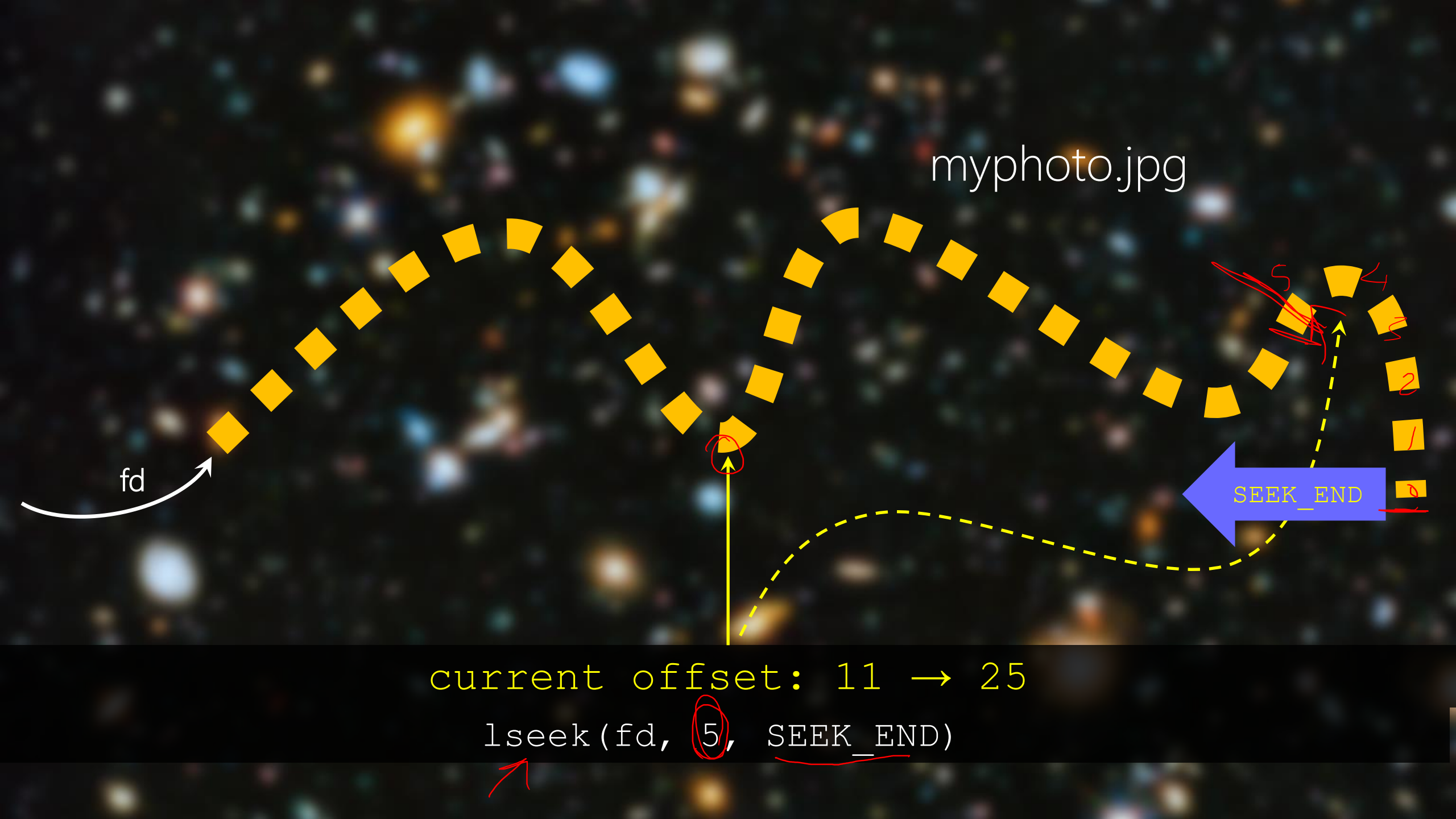
```
#include <unistd.h>
int lseek(int fd, off_t offset, int whence);
```

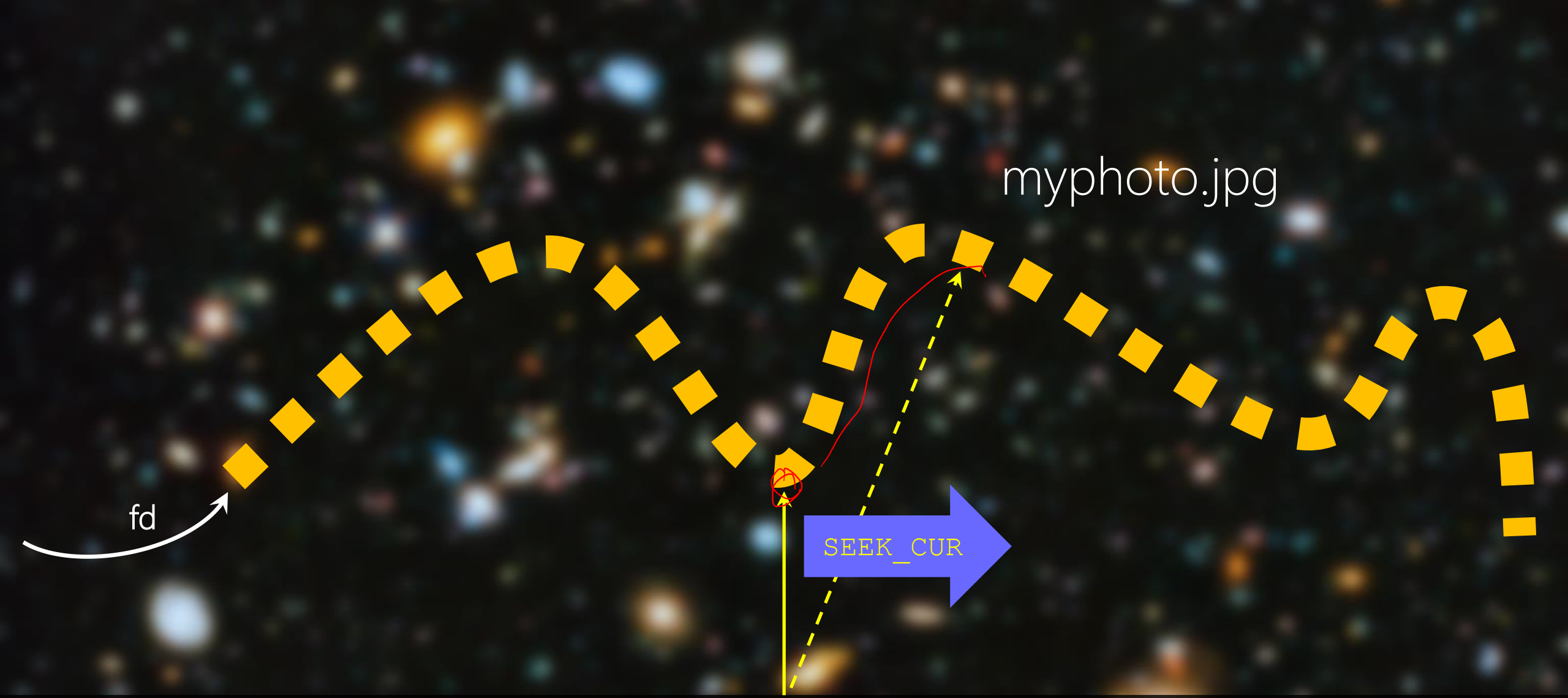
file's new offset if OK (can be negative)

−1 on error

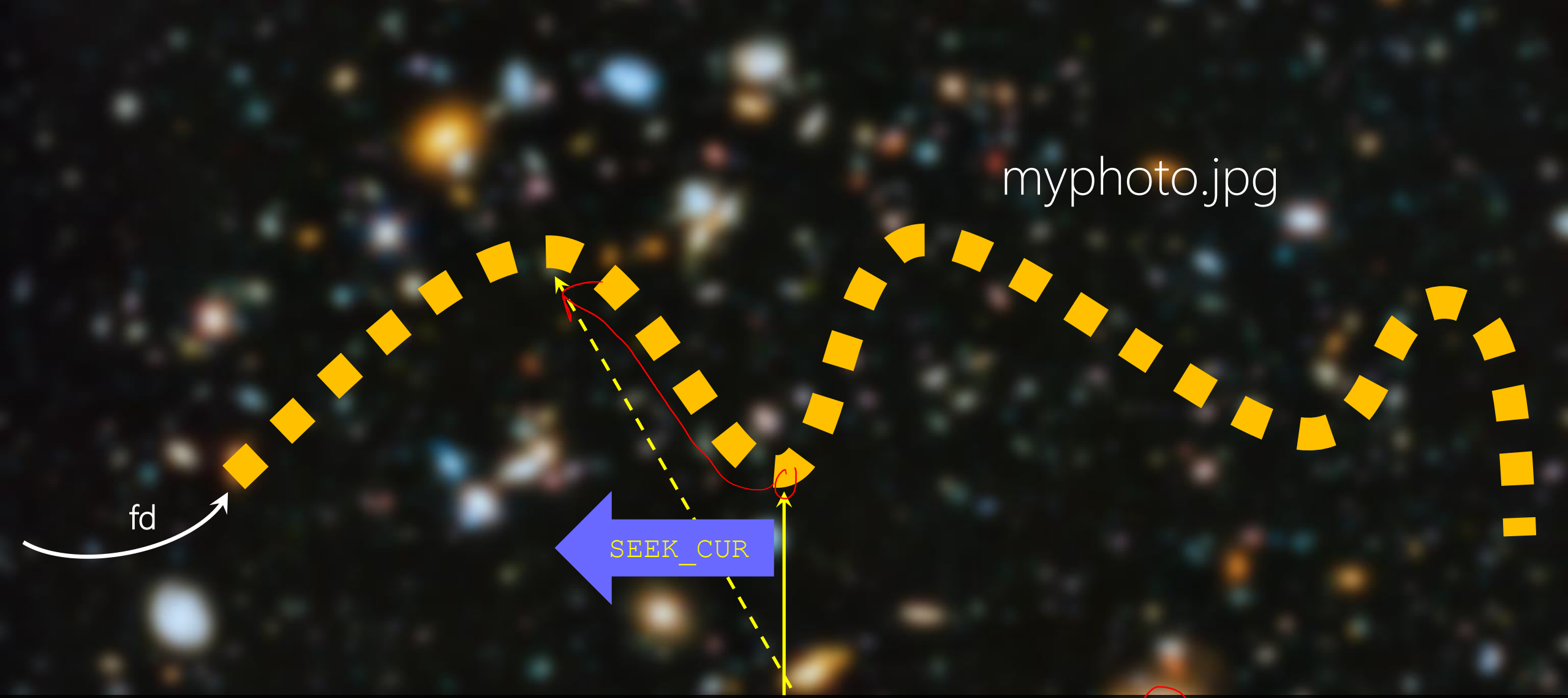

myphoto.jpg
fd
SEEK_SET
SEEK_CUR
SEEK_CUR
SEEK_END
current offset: 11


myphoto.jpg
fd
SEEK_SET
lseek(fd, 5, SEEK_SET)
current offset: 11 → 5


myphoto.jpg
fd
SEEK_END
current offset: 11 → 25
lseek(fd, 5, SEEK_END)




```
current offset: 11 → +5 = 16
lseek(fd, 5, SEEK_CUR)
```



current offset: 11 → -5 = 6

```
lseek(fd, -5, SEEK_CUR)
```

lseek

New current offset

```
#include <unistd.h>
int lseek(int fd, off_t offset, int whence);
```

file's new offset if OK (can be negative)

−1 on error

lseek

How to know the value of current offset?

lseek

How to know the value of current offset?

```
long cur_offset;
cur_offset = lseek(fd, 0, SEEK_CUR);
```

lseek

How to know the file can be seekable?

```
long cur_offset;
cur_offset = lseek(fd, 0, SEEK_CUR);
if (cur_offset == -1){
        /* either the fd is on a file which is not seekable
           or an error occurred /*
}
```