

# Informática II

## Manejo de archivos en lenguaje C (GNU/Linux)

Gonzalo F. Perez Paina



Universidad Tecnológica Nacional  
Facultad Regional Córdoba  
UTN-FRC

– 2019 –

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

```
/--  
|-- bin  
|-- boot  
|-- dev  
|-- etc  
|-- home  
|-- lib  
|-- media  
|-- mnt  
|-- opt  
|-- proc  
|-- root  
|-- sys  
|-- usr  
|-- var
```

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

```
/--
|-- bin
|-- boot
|-- dev
|-- etc
|-- home
|-- lib
|-- media
|-- mnt
|-- opt
|-- proc
|-- root
|-- sys
|-- usr
|-- var
```

► /bin: ejecutables (binarios) de los comandos básicos

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

- /--
  - |-- bin
  - |-- boot
  - |-- dev
  - |-- etc
  - |-- home
  - |-- lib
  - |-- media
  - |-- mnt
  - |-- opt
  - |-- proc
  - |-- root
  - |-- sys
  - |-- usr
  - |-- var
- ▶ /bin: ejecutables (binarios) de los comandos básicos
  - ▶ /boot: cargador de arranque (boot loader), Kernel, archivos `initrd`

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

<code>/--</code>	▶ <code>/bin</code> : ejecutables (binarios) de los comandos básicos
<code> -- bin</code>	▶ <code>/boot</code> : cargador de arranque (boot loader), Kernel, archivos <code>initrd</code>
<code> -- boot</code>	
<code> -- dev</code>	▶ <code>/etc</code> : archivos de configuración del sistema
<code> -- etc</code>	
<code> -- home</code>	
<code> -- lib</code>	
<code> -- media</code>	
<code> -- mnt</code>	
<code> -- opt</code>	
<code> -- proc</code>	
<code> -- root</code>	
<code> -- sys</code>	
<code> -- usr</code>	
<code> -- var</code>	

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

/--	▶ /bin: ejecutables (binarios) de los comandos básicos
-- bin	▶ /boot: cargador de arranque (boot loader), Kernel, archivos <code>initrd</code>
-- boot	
-- dev	▶ /etc: archivos de configuración del sistema
-- etc	
-- home	▶ /home: directorios de usuarios (/home/gonzalo)
-- lib	
-- media	
-- mnt	
-- opt	
-- proc	
-- root	
-- sys	
-- usr	
-- var	

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

/--	▶ /bin: ejecutables (binarios) de los comandos básicos
-- bin	▶ /boot: cargador de arranque (boot loader), Kernel, archivos <code>initrd</code>
-- boot	
-- dev	▶ /etc: archivos de configuración del sistema
-- etc	▶ /home: directorios de usuarios ( <code>/home/gonzalo</code> )
-- home	▶ /mnt: sistemas de archivos montados temporalmente
-- lib	
-- media	
-- mnt	
-- opt	
-- proc	
-- root	
-- sys	
-- usr	
-- var	

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

<code>/--</code>	▶ <code>/bin</code> : ejecutables (binarios) de los comandos básicos
<code> -- bin</code>	▶ <code>/boot</code> : cargador de arranque (boot loader), Kernel, archivos <code>initrd</code>
<code> -- boot</code>	
<code> -- dev</code>	▶ <code>/etc</code> : archivos de configuración del sistema
<code> -- etc</code>	▶ <code>/home</code> : directorios de usuarios ( <code>/home/gonzalo</code> )
<code> -- home</code>	▶ <code>/mnt</code> : sistemas de archivos montados temporalmente
<code> -- lib</code>	▶ <code>/usr</code> : jerarquía secundaria para datos compartidos (read-only)
<code> -- media</code>	
<code> -- mnt</code>	
<code> -- opt</code>	
<code> -- proc</code>	
<code> -- root</code>	
<code> -- sys</code>	
<code> -- usr</code>	
<code> -- var</code>	

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

<code>/--</code>	▶ <code>/bin</code> : ejecutables (binarios) de los comandos básicos
<code> -- bin</code>	▶ <code>/boot</code> : cargador de arranque (boot loader), Kernel, archivos <code>initrd</code>
<code> -- boot</code>	
<code> -- dev</code>	▶ <code>/etc</code> : archivos de configuración del sistema
<code> -- etc</code>	▶ <code>/home</code> : directorios de usuarios ( <code>/home/gonzalo</code> )
<code> -- home</code>	▶ <code>/mnt</code> : sistemas de archivos montados temporalmente
<code> -- lib</code>	▶ <code>/usr</code> : jerarquía secundaria para datos compartidos (read-only)
<code> -- media</code>	
<code> -- mnt</code>	▶ <code>/opt</code> : complementos de paquetes de software de aplicación
<code> -- opt</code>	
<code> -- proc</code>	
<code> -- root</code>	
<code> -- sys</code>	
<code> -- usr</code>	
<code> -- var</code>	

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

<code>/--</code>	▶ <code>/bin</code> : ejecutables (binarios) de los comandos básicos
<code> -- bin</code>	▶ <code>/boot</code> : cargador de arranque (boot loader), Kernel, archivos <code>initrd</code>
<code> -- boot</code>	
<code> -- dev</code>	▶ <code>/etc</code> : archivos de configuración del sistema
<code> -- etc</code>	▶ <code>/home</code> : directorios de usuarios ( <code>/home/gonzalo</code> )
<code> -- home</code>	▶ <code>/mnt</code> : sistemas de archivos montados temporalmente
<code> -- lib</code>	▶ <code>/usr</code> : jerarquía secundaria para datos compartidos (read-only)
<code> -- media</code>	
<code> -- mnt</code>	▶ <code>/opt</code> : complementos de paquetes de software de aplicación
<code> -- opt</code>	
<code> -- proc</code>	▶ <code>/proc</code> : sistema de archivo virtual, estado de procesos
<code> -- root</code>	
<code> -- sys</code>	
<code> -- usr</code>	
<code> -- var</code>	

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

<code>/--</code>	▶ <code>/bin</code> : ejecutables (binarios) de los comandos básicos
<code> -- bin</code>	▶ <code>/boot</code> : cargador de arranque (boot loader), Kernel, archivos <code>initrd</code>
<code> -- boot</code>	
<code> -- dev</code>	▶ <code>/etc</code> : archivos de configuración del sistema
<code> -- etc</code>	▶ <code>/home</code> : directorios de usuarios ( <code>/home/gonzalo</code> )
<code> -- home</code>	▶ <code>/mnt</code> : sistemas de archivos montados temporalmente
<code> -- lib</code>	▶ <code>/usr</code> : jerarquía secundaria para datos compartidos (read-only)
<code> -- media</code>	
<code> -- mnt</code>	▶ <code>/opt</code> : complementos de paquetes de software de aplicación
<code> -- opt</code>	
<code> -- proc</code>	▶ <code>/proc</code> : sistema de archivo virtual, estado de procesos
<code> -- root</code>	▶ <code>/sys</code> : sistema de archivos para objetos exportados del Kernel
<code> -- sys</code>	
<code> -- usr</code>	
<code> -- var</code>	

# Manejo de archivos en C

Estándar de la jerarquía del sistema de archivo (FHS: Filesystem Hierarchy Standard) define los directorios principales y su contenido en los sistemas Linux.

- |                        |   |
|------------------------|---|
| <code>/--</code>       | ▶ <code>/bin</code> : ejecutables (binarios) de los comandos básicos                            |
| <code> -- bin</code>   | ▶ <code>/boot</code> : cargador de arranque (boot loader), Kernel, archivos <code>initrd</code> |
| <code> -- boot</code>  |   |
| <code> -- dev</code>   | ▶ <code>/etc</code> : archivos de configuración del sistema                                     |
| <code> -- etc</code>   | ▶ <code>/home</code> : directorios de usuarios ( <code>/home/gonzalo</code> )                   |
| <code> -- home</code>  | ▶ <code>/mnt</code> : sistemas de archivos montados temporalmente                               |
| <code> -- lib</code>   | ▶ <code>/usr</code> : jerarquía secundaria para datos compartidos (read-only)                   |
| <code> -- media</code> |   |
| <code> -- mnt</code>   | ▶ <code>/opt</code> : complementos de paquetes de software de aplicación                        |
| <code> -- opt</code>   |   |
| <code> -- proc</code>  | ▶ <code>/proc</code> : sistema de archivo virtual, estado de procesos                           |
| <code> -- root</code>  | ▶ <code>/sys</code> : sistema de archivos para objetos exportados del Kernel                    |
| <code> -- sys</code>   |   |
| <code> -- usr</code>   |   |
| <code> -- var</code>   | ▶ <code>/dev</code> : archivos de dispositivos (caracteres o bloques)                           |

# Manejo de archivos en C

Cada archivo tiene un nombre y algunas propiedades como la fecha de creación/modificación, permisos, etc.



# Manejo de archivos en C

- ▶ Cada programa en ejecución (proceso) tiene una cantidad de archivos asociados (descriptor de archivo).

# Manejo de archivos en C

- ▶ Cada programa en ejecución (proceso) tiene una cantidad de archivos asociados (descriptor de archivo).
- ▶ **Descriptores de archivos:** son números enteros pequeños que se pueden utilizar para acceder a estos archivos o dispositivos.

# Manejo de archivos en C

- ▶ Cada programa en ejecución (proceso) tiene una cantidad de archivos asociados (descriptor de archivo).
- ▶ **Descriptores de archivos:** son números enteros pequeños que se pueden utilizar para acceder a estos archivos o dispositivos.

Todo programa tiene abierto tres descriptores de archivos:

- ▶ 0: entrada estándar (`stdin`)
- ▶ 1: salida estándar (`stdout`)
- ▶ 2: error estándar (`stderr`)

# Manejo de archivos en C – Ejemplos

escritura.c

---

```
1 #include <unistd.h>
2
3 int main(void)
4 {
5     if((write(1, "Hola mundo!\n", 12)) != 12)
6         write(2, "ERROR\n", 6);
7
8     return 0;
9 }
```

---

# Manejo de archivos en C – Ejemplos

lectura.c

---

```
1 #include <unistd.h>
2
3 int main(void) {
4     char buffer[128];
5     int nread;
6
7     nread = read(0, buffer, 128);
8     if(nread == -1)
9         write(2, "Error de lectura\n", 17);
10
11     if((write(1, buffer, nread)) != nread)
12         write(2, "Error de escritura\n", 19);
13
14     return 0;
15 }
```

---

# Manejo de archivos en C – Ejemplos

## lectura.c

---

```
1 #include <unistd.h>
2
3 int main(void) {
4     char buffer[128];
5     int nread;
6
7     nread = read(0, buffer, 128);
8     if(nread == -1)
9         write(2, "Error de lectura\n", 17);
10
11     if((write(1, buffer, nread)) != nread)
12         write(2, "Error de escritura\n", 19);
13
14     return 0;
15 }
```

---

- Compilar y ejecutar (con/sin redirección)

# Manejo de archivos en C – Ejemplos

## lectura.c

---

```
1 #include <unistd.h>
2
3 int main(void) {
4     char buffer[128];
5     int nread;
6
7     nread = read(0, buffer, 128);
8     if(nread == -1)
9         write(2, "Error de lectura\n", 17);
10
11     if((write(1, buffer, nread)) != nread)
12         write(2, "Error de escritura\n", 19);
13
14     return 0;
15 }
```

---

- ▶ Compilar y ejecutar (con/sin redirección)
- ▶ Ejecutar con redirección  
> ./a.out < prueba.txt

# Manejo de archivos en C – Ejemplos

## lectura.c

---

```
1 #include <unistd.h>
2
3 int main(void) {
4     char buffer[128];
5     int nread;
6
7     nread = read(0, buffer, 128);
8     if(nread == -1)
9         write(2, "Error de lectura\n", 17);
10
11     if((write(1, buffer, nread)) != nread)
12         write(2, "Error de escritura\n", 19);
13
14     return 0;
15 }
```

---

- ▶ Compilar y ejecutar (con/sin redirección)
- ▶ Ejecutar con redirección  
> ./a.out < prueba.txt
- ▶ Ver número de proceso  
> ps aux | grep a.out

# Manejo de archivos en C – Ejemplos

## lectura.c

---

```
1 #include <unistd.h>
2
3 int main(void) {
4     char buffer[128];
5     int nread;
6
7     nread = read(0, buffer, 128);
8     if(nread == -1)
9         write(2, "Error de lectura\n", 17);
10
11     if((write(1, buffer, nread)) != nread)
12         write(2, "Error de escritura\n", 19);
13
14     return 0;
15 }
```

---

- ▶ Compilar y ejecutar (con/sin redirección)
- ▶ Ejecutar con redirección  
> ./a.out < prueba.txt
- ▶ Ver número de proceso  
> ps aux | grep a.out
- ▶ Ver directorio /proc/<PID>

# Manejo de archivos en C – Ejemplos

## lectura.c

---

```
1 #include <unistd.h>
2
3 int main(void) {
4     char buffer[128];
5     int nread;
6
7     nread = read(0, buffer, 128);
8     if(nread == -1)
9         write(2, "Error de lectura\n", 17);
10
11     if((write(1, buffer, nread)) != nread)
12         write(2, "Error de escritura\n", 19);
13
14     return 0;
15 }
```

---

- ▶ Compilar y ejecutar (con/sin redirección)
- ▶ Ejecutar con redirección  
> ./a.out < prueba.txt
- ▶ Ver número de proceso  
> ps aux | grep a.out
- ▶ Ver directorio /proc/<PID>
- ▶ Ver /proc/<PID>/fd

# Manejo de archivos en C – Ejemplos

## lectura.c

---

```
1 #include <unistd.h>
2
3 int main(void) {
4     char buffer[128];
5     int nread;
6
7     nread = read(0, buffer, 128);
8     if(nread == -1)
9         write(2, "Error de lectura\n", 17);
10
11     if((write(1, buffer, nread)) != nread)
12         write(2, "Error de escritura\n", 19);
13
14     return 0;
15 }
```

---

- ▶ Compilar y ejecutar (con/sin redirección)
- ▶ Ejecutar con redirección  
> ./a.out < prueba.txt
- ▶ Ver número de proceso  
> ps aux | grep a.out
- ▶ Ver directorio /proc/<PID>
- ▶ Ver /proc/<PID>/fd
- ▶ Desde otra terminal  
> echo "Hola" >  
/proc/<PID>/fd/1

# Manejo de archivos en C

Los programas pueden manejar archivos de disco, puerto serie, y otros dispositivos (excepto conexiones de red), todos de la misma forma.

# Manejo de archivos en C

Los programas pueden manejar archivos de disco, puerto serie, y otros dispositivos (excepto conexiones de red), todos de la misma forma.

## Archivos de dispositivos

# Manejo de archivos en C

Los programas pueden manejar archivos de disco, puerto serie, y otros dispositivos (excepto conexiones de red), todos de la misma forma.

## Archivos de dispositivos

- ▶ Los dispositivos de hardware se representan por archivos (mapean).

# Manejo de archivos en C

Los programas pueden manejar archivos de disco, puerto serie, y otros dispositivos (excepto conexiones de red), todos de la misma forma.

## Archivos de dispositivos

- ▶ Los dispositivos de hardware se representan por archivos (mapean).
- ▶ Los dispositivos se clasifican en: dispositivos de **caracteres** o de **bloques**.

# Manejo de archivos en C

Los programas pueden manejar archivos de disco, puerto serie, y otros dispositivos (excepto conexiones de red), todos de la misma forma.

## Archivos de dispositivos

- ▶ Los dispositivos de hardware se representan por archivos (mapean).
- ▶ Los dispositivos se clasifican en: dispositivos de **caracteres** o de **bloques**.

```
> ls -l /dev
crw----- 1 root root      5,   1 jul 10 13:55 console
crw-rw-rw- 1 root root      1,   3 jul 10 13:54 null
```

# Manejo de archivos en C

Los programas pueden manejar archivos de disco, puerto serie, y otros dispositivos (excepto conexiones de red), todos de la misma forma.

## Archivos de dispositivos

- ▶ Los dispositivos de hardware se representan por archivos (mapean).
- ▶ Los dispositivos se clasifican en: dispositivos de **caracteres** o de **bloques**.

```
> ls -l /dev
crw----- 1 root root      5,   1 jul 10 13:55 console
crw-rw-rw- 1 root root      1,   3 jul 10 13:54 null
brw-rw---- 1 root disk     8,   0 jul 19 13:51 sda
brw-rw---- 1 root disk     8,   1 jul 19 13:51 sda1
brw-rw---- 1 root disk     8,   2 jul 19 13:51 sda2
brw-rw---- 1 root disk     8,   3 jul 19 13:51 sda3
```

# Manejo de archivos en C

Los programas pueden manejar archivos de disco, puerto serie, y otros dispositivos (excepto conexiones de red), todos de la misma forma.

## Archivos de dispositivos

- ▶ Los dispositivos de hardware se representan por archivos (mapean).
- ▶ Los dispositivos se clasifican en: dispositivos de **caracteres** o de **bloques**.

```
> ls -l /dev
crw----- 1 root root      5,   1 jul 10 13:55 console
crw-rw-rw- 1 root root      1,   3 jul 10 13:54 null
brw-rw---- 1 root disk     8,   0 jul 19 13:51 sda
brw-rw---- 1 root disk     8,   1 jul 19 13:51 sda1
brw-rw---- 1 root disk     8,   2 jul 19 13:51 sda2
brw-rw---- 1 root disk     8,   3 jul 19 13:51 sda3
lrwxrwxrwx 1 root root    15 jul 10 13:53 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root    15 jul 10 13:53 stdin  -> /proc/self/fd/0
lrwxrwxrwx 1 root root    15 jul 10 13:53 stdout -> /proc/self/fd/1
```

# Manejo de archivos en C

Los programas pueden manejar archivos de disco, puerto serie, y otros dispositivos (excepto conexiones de red), todos de la misma forma.

## Archivos de dispositivos

- ▶ Los dispositivos de hardware se representan por archivos (mapean).
- ▶ Los dispositivos se clasifican en: dispositivos de **caracteres** o de **bloques**.

```
> ls -l /dev
crw----- 1 root root      5,   1 jul 10 13:55 console
crw-rw-rw- 1 root root      1,   3 jul 10 13:54 null
brw-rw---- 1 root disk     8,   0 jul 19 13:51 sda
brw-rw---- 1 root disk     8,   1 jul 19 13:51 sda1
brw-rw---- 1 root disk     8,   2 jul 19 13:51 sda2
brw-rw---- 1 root disk     8,   3 jul 19 13:51 sda3
lrwxrwxrwx 1 root root      15 jul 10 13:53 stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root      15 jul 10 13:53 stdin  -> /proc/self/fd/0
lrwxrwxrwx 1 root root      15 jul 10 13:53 stdout -> /proc/self/fd/1
crw-rw---- 1 root dialout  4,  64 jul 10 13:53 ttyS0
crw-rw---- 1 root dialout 188,  0 jul 19 19:10 ttyUSB0
```

# Manejo de archivos en C

## Acceso a archivos

- ▶ Funciones de bajo nivel (llamada al sistema y drives de dispositivos)
- ▶ Funciones de alto nivel (biblioteca de entrada/salida) [buffers]

# Manejo de archivos en C

## Acceso a archivos

- ▶ Funciones de bajo nivel (llamada al sistema y drives de dispositivos)
  - ▶ Funciones de alto nivel (biblioteca de entrada/salida) [buffers]
- 
- ▶ **Bajo nivel:** Archivo de cabecera `unistd.h` – En los lenguajes C y C++ este archivo de cabecera define la API que brinda acceso al sistema operativo POSIX (descriptor de archivo).

# Manejo de archivos en C

## Acceso a archivos

- ▶ Funciones de bajo nivel (llamada al sistema y drives de dispositivos)
- ▶ Funciones de alto nivel (biblioteca de entrada/salida) [buffers]
  
- ▶ **Bajo nivel:** Archivo de cabecera `unistd.h` – En los lenguajes C y C++ este archivo de cabecera define la API que brinda acceso al sistema operativo POSIX (descriptor de archivo).  
(`open`, `close`, `write`, `read`, etc.)

# Manejo de archivos en C

## Acceso a archivos

- ▶ Funciones de bajo nivel (llamada al sistema y drives de dispositivos)
- ▶ Funciones de alto nivel (biblioteca de entrada/salida) [buffers]
  
- ▶ **Bajo nivel:** Archivo de cabecera `unistd.h` – En los lenguajes C y C++ este archivo de cabecera define la API que brinda acceso al sistema operativo POSIX (descriptor de archivo). (`open`, `close`, `write`, `read`, etc.)
- ▶ **Alto nivel:** Archivo de cabecera `stdio.h` (ANSI C) – Manejo de *stream*, se implementa como puntero a estructura de tipo `FILE*`.

# Manejo de archivos en C

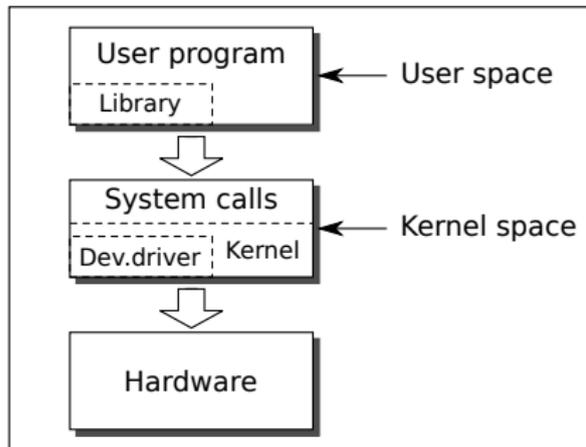
## Acceso a archivos

- ▶ Funciones de bajo nivel (llamada al sistema y drives de dispositivos)
  - ▶ Funciones de alto nivel (biblioteca de entrada/salida) [buffers]
- 
- ▶ **Bajo nivel:** Archivo de cabecera `unistd.h` – En los lenguajes C y C++ este archivo de cabecera define la API que brinda acceso al sistema operativo POSIX (descriptor de archivo).  
(`open`, `close`, `write`, `read`, etc.)
  - ▶ **Alto nivel:** Archivo de cabecera `stdio.h` (ANSI C) – Manejo de *stream*, se implementa como puntero a estructura de tipo `FILE*`.  
(`fopen`, `fclose`, `fwrite`, `fread`, etc.)

# Manejo de archivos en C

## Acceso a archivos

- ▶ Funciones de bajo nivel (llamada al sistema y drives de dispositivos)
  - ▶ Funciones de alto nivel (biblioteca de entrada/salida) [buffers]
- 
- ▶ **Bajo nivel:** Archivo de cabecera `unistd.h` – En los lenguajes C y C++ este archivo de cabecera define la API que brinda acceso al sistema operativo POSIX (descriptor de archivo). (`open`, `close`, `write`, `read`, etc.)
  - ▶ **Alto nivel:** Archivo de cabecera `stdio.h` (ANSI C) – Manejo de *stream*, se implementa como puntero a estructura de tipo `FILE*`. (`fopen`, `fclose`, `fwrite`, `fread`, etc.)





# Manejo de archivos en C – Funciones de bajo nivel

Se utilizan cinco funciones: `open`, `close`, `write`, `read`, e `ioctl`.

# Manejo de archivos en C – Funciones de bajo nivel

Se utilizan cinco funciones: `open`, `close`, `write`, `read`, e `ioctl`.

- ▶ `open()`: Abrir archivo o dispositivo

# Manejo de archivos en C – Funciones de bajo nivel

Se utilizan cinco funciones: `open`, `close`, `write`, `read`, e `ioctl`.

- ▶ `open()`: Abrir archivo o dispositivo
- ▶ `close()`: Cerrar archivo o dispositivo

# Manejo de archivos en C – Funciones de bajo nivel

Se utilizan cinco funciones: `open`, `close`, `write`, `read`, e `ioctl`.

- ▶ `open()`: Abrir archivo o dispositivo
- ▶ `close()`: Cerrar archivo o dispositivo
- ▶ `read()`: Leer archivo o dispositivo

# Manejo de archivos en C – Funciones de bajo nivel

Se utilizan cinco funciones: `open`, `close`, `write`, `read`, e `ioctl`.

- ▶ `open()`: Abrir archivo o dispositivo
- ▶ `close()`: Cerrar archivo o dispositivo
- ▶ `read()`: Leer archivo o dispositivo
- ▶ `write()`: Escribir archivo o dispositivo

# Manejo de archivos en C – Funciones de bajo nivel

Se utilizan cinco funciones: `open`, `close`, `write`, `read`, e `ioctl`.

- ▶ `open()`: Abrir archivo o dispositivo
- ▶ `close()`: Cerrar archivo o dispositivo
- ▶ `read()`: Leer archivo o dispositivo
- ▶ `write()`: Escribir archivo o dispositivo
- ▶ `ioctl()`: Intercambiar información de control con el driver

# Manejo de archivos en C – Funciones de bajo nivel

Se utilizan cinco funciones: `open`, `close`, `write`, `read`, e `ioctl`.

- ▶ `open()`: Abrir archivo o dispositivo
- ▶ `close()`: Cerrar archivo o dispositivo
- ▶ `read()`: Leer archivo o dispositivo
- ▶ `write()`: Escribir archivo o dispositivo
- ▶ `ioctl()`: Intercambiar información de control con el driver

En el Kernel están los *drivers de dispositivos* (device drivers): interfaz de bajo nivel para el control de hardware.

# Manejo de archivos en C – Funciones de bajo nivel

Se utilizan cinco funciones: `open`, `close`, `write`, `read`, e `ioctl`.

- ▶ `open()`: Abrir archivo o dispositivo
- ▶ `close()`: Cerrar archivo o dispositivo
- ▶ `read()`: Leer archivo o dispositivo
- ▶ `write()`: Escribir archivo o dispositivo
- ▶ `ioctl()`: Intercambiar información de control con el driver

En el Kernel están los *drivers de dispositivos* (device drivers): interfaz de bajo nivel para el control de hardware.

Descriptores de archivos abiertos en cualquier programa: 0, 1 y 2

# Manejo de archivos en C – Funciones de bajo nivel

## Abrir archivo

---

```
#include <fcntl.h> /* File control definitions */
#include <unistd.h> /* UNIX standard function definitions */

int open(const char* path, int oflags);
```

---

- ▶ **path**: nombre del archivo o dispositivo
- ▶ **oflags**: indica acciones al abrir el archivo (O\_RDONLY, O\_WRONLY, O\_RDWR)

Devuelve el *descriptor de archivo* (entero no negativo) si tuvo éxito, o -1 si falló.

# Manejo de archivos en C – Funciones de bajo nivel

## Abrir archivo

---

```
#include <fcntl.h> /* File control definitions */
#include <unistd.h> /* UNIX standard function definitions */

int open(const char* path, int oflags);
```

---

- ▶ **path**: nombre del archivo o dispositivo
- ▶ **oflags**: indica acciones al abrir el archivo (O\_RDONLY, O\_WRONLY, O\_RDWR)

Devuelve el *descriptor de archivo* (entero no negativo) si tuvo éxito, o -1 si falló.

## Cerrar archivo

---

```
#include <fcntl.h> /* File control definitions */
#include <unistd.h> /* UNIX standard function definitions */

int close(int fildes);
```

---

- ▶ **fildes**: descriptor de archivo

# Manejo de archivos en C – Funciones de bajo nivel

## Leer archivo

```
size_t read(int fildes, void *buf, size_t nbytes);
```

## Escribir archivo

```
size_t write(int fildes, const void *buf, size_t nbytes);
```

## I/O control

```
int ioctl(int fildes, int cmd, ...);
```

- ▶ **buf**: Buffer para la lectura/escritura
- ▶ **nbytes**: Cantidad de bytes a leer/escribir
- ▶ **cmd**: Acción a realizar sobre el archivo



# Manejo de archivos en C – Funciones de alto nivel

Algunas de las funciones de alto nivel son:

# Manejo de archivos en C – Funciones de alto nivel

Algunas de las funciones de alto nivel son:

- ▶ `fopen`, `fclose`

# Manejo de archivos en C – Funciones de alto nivel

Algunas de las funciones de alto nivel son:

- ▶ `fopen`, `fclose`
- ▶ `fwrite`, `fread`

# Manejo de archivos en C – Funciones de alto nivel

Algunas de las funciones de alto nivel son:

- ▶ `fopen`, `fclose`
- ▶ `fwrite`, `fread`
- ▶ `fflush`, `fseek`

# Manejo de archivos en C – Funciones de alto nivel

Algunas de las funciones de alto nivel son:

- ▶ `fopen, fclose`
- ▶ `fwrite, fread`
- ▶ `fflush, fseek`
- ▶ `fputs, fgets`

# Manejo de archivos en C – Funciones de alto nivel

Algunas de las funciones de alto nivel son:

- ▶ `fopen, fclose`
- ▶ `fwrite, fread`
- ▶ `fflush, fseek`
- ▶ `fputs, fgets`
- ▶ `fscanf, fprintf`

# Manejo de archivos en C – Funciones de alto nivel

Algunas de las funciones de alto nivel son:

- ▶ `fopen, fclose`
- ▶ `fwrite, fread`
- ▶ `fflush, fseek`
- ▶ `fputs, fgets`
- ▶ `fscanf, fprintf`
- ▶ `getline, getdelim`

# Manejo de archivos en C – Funciones de alto nivel

Algunas de las funciones de alto nivel son:

- ▶ `fopen`, `fclose`
- ▶ `fwrite`, `fread`
- ▶ `fflush`, `fseek`
- ▶ `fputs`, `fgets`
- ▶ `fscanf`, `fprintf`
- ▶ `getline`, `getdelim`

Flujos de datos abiertos en cualquier programa: `stdin`, `stdout` y `stderr`

# Manejo de archivos en C – Funciones de alto nivel

## Abrir archivo

---

```
#include <stdio.h>

FILE *fopen(const char *filename, const char *mode);
```

---

- ▶ **filename**: nombre del archivo o dispositivo
- ▶ **mode**: indica abrir el archivo ("w", "r", "w+", "r+", "a", "a+b", etc.)

Devuelve un puntero (no nulo) **\*FILE** si tuvo éxito, o **NULL** si falló.

# Manejo de archivos en C – Funciones de alto nivel

## Abrir archivo

---

```
#include <stdio.h>

FILE *fopen(const char *filename, const char *mode);
```

---

- ▶ **filename**: nombre del archivo o dispositivo
- ▶ **mode**: indica abrir el archivo ("w", "r", "w+", "r+", "a", "a+b", etc.)

Devuelve un puntero (no nulo) **\*FILE** si tuvo éxito, o **NULL** si falló.

## Cerrar archivo

---

```
#include <stdio.h>

int fclose(FILE *stream);
```

---

- ▶ **sream**: flujo de datos a cerrar

# Manejo de archivos en C – Funciones de alto nivel

## Leer archivo

```
size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
```

## Escribir archivo

```
size_t fwrite(const void *ptr, size_t size, size_t nitems, FILE *stream);
```

- ▶ `ptr`: Puntero al buffer de datos para la lectura/escritura
  - ▶ `size`: Tamaño del registro a transferir
  - ▶ `nitems`: Cantidad de registros a transferir
-

# Manejo de archivos en C – Funciones de alto nivel

## Leer archivo

```
size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
```

## Escribir archivo

```
size_t fwrite(const void *ptr, size_t size, size_t nitems, FILE *stream);
```

- ▶ **ptr**: Puntero al buffer de datos para la lectura/escritura
  - ▶ **size**: Tamaño del registro a transferir
  - ▶ **nitems**: Cantidad de registros a transferir
- 

## Leer cadena desde un archivo

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

- ▶ **lineptr**: Buffer para almacenar la cadena
- ▶ **n**: Longitud de la cadena leída

# Manejo de archivos en C – Funciones de alto nivel

## Flush

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

Fuerza que los datos del buffer sean efectivamente enviados.

## Seek

```
int fseek (FILE *stream, long int offset, int whence);
```

Fija el puntero del `stream` para la próxima transferencia.

- ▶ `offset`: posición medida en bytes
- ▶ `whence`: puede ser
  1. `SEEK_SET`: posición. absoluta
  2. `SEEK_CUR`: posición. actual
  3. `SEEK_END`: relativo al final

