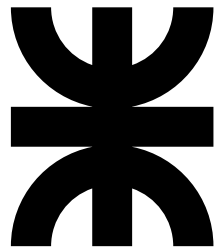
Informática II

Programación en lenguaje C en Linux

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– 2017 –

.

Programación en lenguaje C en Linux

Herramienta de compilación

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

Programación en lenguaje C en Linux
Herramienta de compilación

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.

Programación en lenguaje C en Linux
Herramienta de compilación

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++

Programación en lenguaje C en Linux
Herramienta de compilación

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++
- Realiza la optimización del código

Programación en lenguaje C en Linux
Herramienta de compilación

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++
- Realiza la optimización del código
- Genera información de depuración (debugging)

Programación en lenguaje C en Linux
Herramienta de compilación

Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++
- Realiza la optimización del código
- Genera información de depuración (debugging)
- Es un compilador cruzado (cross-compiler)
  `$ gcc --version`
  `$ gcc -v`
  (`--build`, `--host`, `--target`)

Programación en lenguaje C en Linux
Herramienta de compilación

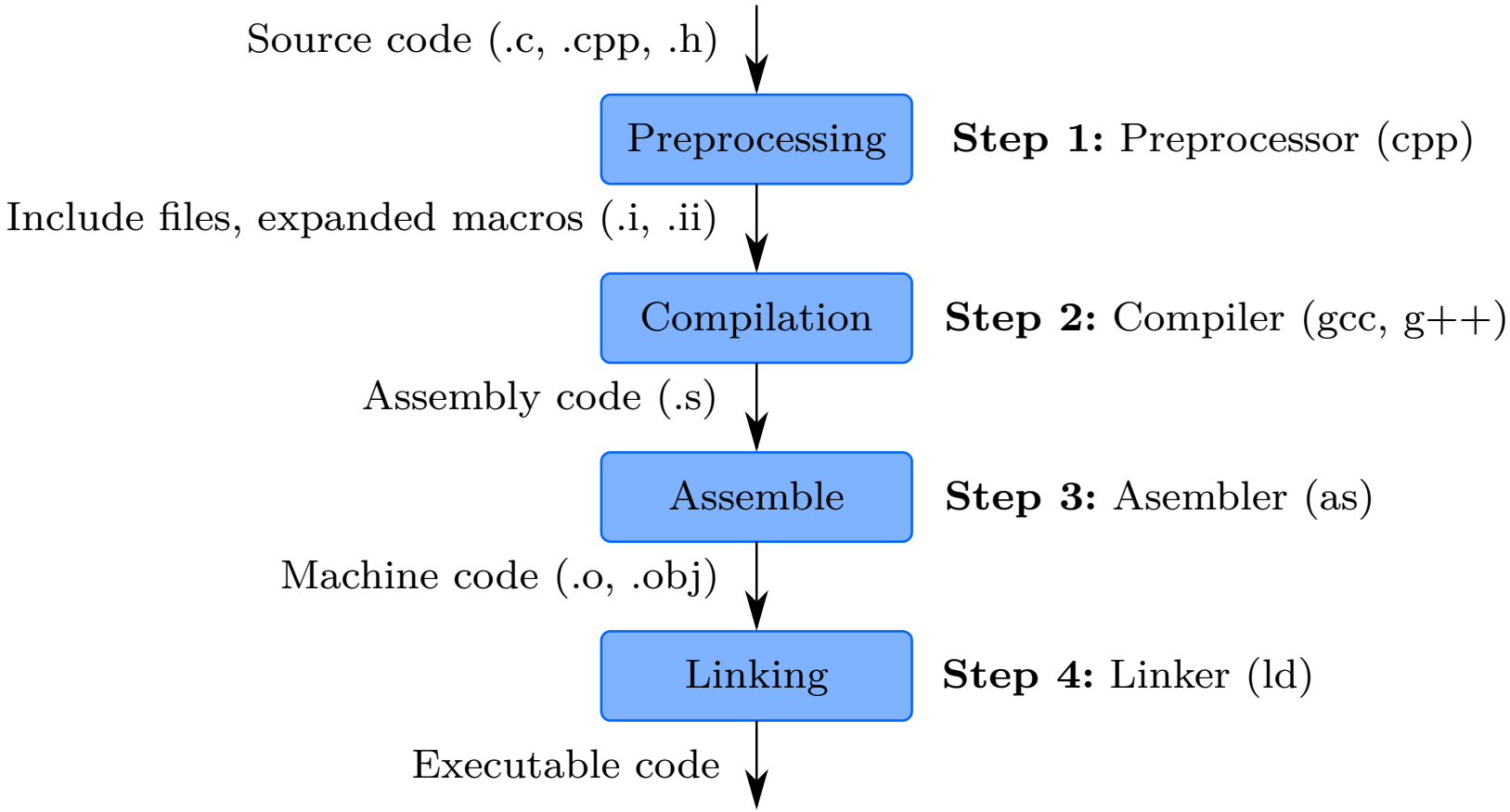
Herramientas de compilación del proyecto GNU: **gcc**, GNU Compiler Collection

- Maneja varios dialectos de C: ANSI-C, tradicional (Kernighan & Ritchie), etc.
- Puede compilar C++
- Realiza la optimización del código
- Genera información de depuración (debugging)
- Es un compilador cruzado (cross-compiler)
  `$ gcc --version`
  `$ gcc -v`
  (`--build`, `--host`, `--target`)

El proceso de compilación/construcción involucra 4 etapas (preprocesamiento, compilación, ensamblado, y linkeo)

Programación en lenguaje C en Linux

Etapas de compilación



Programación en lenguaje C en Linux

Etapas de compilación

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Programación en lenguaje C en Linux

Etapas de compilación

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Compilar

```
$ gcc hello.c
```

Programación en lenguaje C en Linux

Etapas de compilación

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Compilar

```
$ gcc hello.c
```

```
$ gcc hello.c -o hello
```

Programación en lenguaje C en Linux

Etapas de compilación

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Compilar

```
$ gcc hello.c
```

```
$ gcc hello.c -o hello
```

Ejecutar

```
$ ./hello
```

Programación en lenguaje C en Linux

Etapas de compilación

```c
/* Source file: hello.c */
#include <stdio.h>

int main(void)
{
  printf("Hello, Linux programming world!\n");
  return 0;
}
```

Compilar

```
$ gcc hello.c
```

```
$ gcc hello.c -o hello
```

Ejecutar

```
$ ./hello
```

Salida

```
$ Hello Linux programming world!
```

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Preprocessing

```
$ gcc -E hello.c
```

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Preprocessing

```
$ gcc -E hello.c
```

```
$ gcc -E hello.c -o hello.i
```

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Preprocessing

```
$ gcc -E hello.c
```

```
$ gcc -E hello.c -o hello.i
```

```
$ cpp hello.c
```

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Preprocessing

```
$ gcc -E hello.c
```

```
$ gcc -E hello.c -o hello.i
```

```
$ cpp hello.c
```

Copiar el archivo fuente y modificarlo

```
1  /* Hello Linux */
2  #include <stdio.h>
3
4  #define STRING "Hello, Linux programming world!\n"
5
6  int main(void)
7  {
8    /* Using a macro to print a message */
9    printf(STRING);
10   return 0;
11 }
```

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Preprocessing

```
$ gcc -E hello.c
```

```
$ gcc -E hello.c -o hello.i
```

```
$ cpp hello.c
```

Copiar el archivo fuente y modificarlo

```
1  /* Hello Linux */
2  #include <stdio.h>
3
4  #define STRING "Hello, Linux programming world!\n"
5
6  int main(void)
7  {
8    /* Using a macro to print a message */
9    printf(STRING);
10   return 0;
11 }
```

Observar: macros, comentarios, archivos cabecera (includes)

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Linkeo

```
$ gcc hello.o -o hello
```

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Linkeo

```
$ gcc hello.o -o hello
```

(`objdump -x`, `readelf -d`, `ldd`)

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Linkeo

```
$ gcc hello.o -o hello
```

(`objdump -x`, `readelf -d`, `ldd`)

Flag `-save-temps` genera archivos intermedios

```
$ gcc -save-temps hello.c -o hello
```

Programación en lenguaje C en Linux

Etapas de compilación, paso a paso

Compilar (continuando de la etapa anterior)

```
$ gcc -x cpp-output -c hello.i -o hello.o
```

La opción `-x` le indica que continúe desde la etapa indicada (man)

Linkeo

```
$ gcc hello.o -o hello
```

(`objdump -x`, `readelf -d`, `ldd`)

Flag `-save-temps` genera archivos intermedios

```
$ gcc -save-temps hello.c -o hello
```

(Flags `-std=c90`, `-Wall`, `-Werror`)

Archivos de bibliotecas

- `.a`: bibliotecas estáticas
- `.so`: bibliotecas dinámicas

Ejemplos

- `$ gcc -o main main.c /usr/lib/libm.a` (indicando path completo)
- `$ gcc -o main main.c -lm` (con flag `-l`)
-
- `$ gcc -o mainX11 -L/usr/X11/lib mainX11.c -lX11`

Programación en lenguaje C en Linux
Bibliotecas estáticas y dinámicas

Archivos de bibliotecas

- `.a`: bibliotecas estáticas
- `.so`: bibliotecas dinámicas

Ejemplos

- `$ gcc -o main main.c /usr/lib/libm.a` (indicando path completo)
- `$ gcc -o main main.c -lm` (con flag `-l`)
-
- `$ gcc -o mainX11 -L/usr/X11/lib mainX11.c -lX11`

Archivos de cabecera

- `$ gcc -I/usr/include/[subdir] source.c` (directorio `/usr/include` y subdirectorios)

Programación en lenguaje C en Linux

Construcción de bibliotecas estáticas y dinámicas

```c
/* Source file: foo.h */
#ifndef _foo_h
#define _foo_h

void foo(void);

#endif  // _foo_h
```

```c
/* Source file: foo.c */
#include "foo.h"
#include <stdio.h>

void foo(void)
{
  puts("Hello, I'm a library");
}
```

Programación en lenguaje C en Linux

Construcción de bibliotecas estáticas y dinámicas

```c
/* Source file: foo.h */
#ifndef _foo_h
#define _foo_h

void foo(void);

#endif  // _foo_h
```

```c
/* Source file: foo.c */
#include "foo.h"
#include <stdio.h>

void foo(void)
{
  puts("Hello, I'm a library");
}
```

```c
/* Source file: main.c */
#include <stdio.h>
#include "foo.h"

int main(void)
{
  puts("This is a library test...");
  foo();
  return 0;
}
```

Programación en lenguaje C en Linux

Main con parámetros

Cuando se ejecuta un programa en Linux, este comienza en la función **main**.

Declaración de main

```
int main(int argc, char *argv[])
```

Programación en lenguaje C en Linux

Main con parámetros

Cuando se ejecuta un programa en Linux, este comienza en la función **main**.

Declaración de main

```
int main(int argc, char *argv[])
```

Donde:

- **argc**: cantidad de argumentos del programa
- **argv**: vector de cadena de caracteres con estos argumentos

(el valor de retorno por defecto es **int**)

Programación en lenguaje C en Linux

Main con parámetros

Cuando se ejecuta un programa en Linux, este comienza en la función **main**.

Declaración de main

```
int main(int argc, char *argv[])
```

Donde:

- **argc**: cantidad de argumentos del programa
- **argv**: vector de cadena de caracteres con estos argumentos

(el valor de retorno por defecto es **int**)

Los parámetros pasados a **main** provienen de otro programa, generalmente la shell que le ha pedido al SO que ejecute el programa.

Programación en lenguaje C en Linux

Main con parámetros

Cuando se ejecuta un programa en Linux, este comienza en la función **main**.

Declaración de main

```
int main(int argc, char *argv[])
```

Donde:

- **argc**: cantidad de argumentos del programa
- **argv**: vector de cadena de caracteres con estos argumentos

(el valor de retorno por defecto es **int**)

Los parámetros pasados a **main** provienen de otro programa, generalmente la shell que le ha pedido al SO que ejecute el programa.

Programa que imprima el valor de **argc** y **argv**.

Programación en lenguaje C en Linux

Main con parámetros

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int arg;

  for(arg = 0; arg < argc; arg++)
  {
    if(argv[arg][0] == '-')
      printf("option: %s\n", argv[arg]+1);
    else
      printf("argument %d: %s\n", arg, argv[arg]);

  }

  return 0;
}
```

Programación en lenguaje C en Linux

Main con parámetros

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int arg;

  for(arg = 0; arg < argc; arg++)
  {
    if(argv[arg][0] == '-')
      printf("option: %s\n", argv[arg]+1);
    else
      printf("argument %d: %s\n", arg, argv[arg]);

  }

  return 0;
}
```

```
$./main_args -u -s hola -f main.c
argument 0: ./ch04-01_args
option: u
option: s
argument 3: hola
option: f
argument 5: main.c
```

Programación en lenguaje C en Linux

Main con parámetros – `getopt`

La función `getopt()` ayuda a parsear los argumentos de `main`

```
#include <unistd.h>

int getopt(int argc, char *const argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

Programación en lenguaje C en Linux

Main con parámetros – `getopt`

La función `getopt()` ayuda a parsear los argumentos de `main`

```c
#include <unistd.h>

int getopt(int argc, char *const argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

El parámetro `optstring` es una lista de caracteres

- cada uno representa una opción de un único caracter
- si está seguido de dos punto (:), indica que la opción tiene un valor asociado

Programación en lenguaje C en Linux

Main con parámetros – `getopt`

La función **`getopt()`** ayuda a parsear los argumentos de **`main`**

```c
#include <unistd.h>

int getopt(int argc, char *const argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

El parámetro **`optstring`** es una lista de caracteres

- cada uno representa una opción de un único caracter
- si está seguido de dos punto (`:`), indica que la opción tiene un valor asociado

El valor de regreso de **`getopt`** es el caracter de opción siguiente encontrado en el vector **`argv`** (si este existe).

Programación en lenguaje C en Linux

Main con parámetros – `getopt`

La función **`getopt()`** ayuda a parsear los argumentos de **`main`**

```c
#include <unistd.h>

int getopt(int argc, char *const argv[], const char *optstring);
extern char *optarg;
extern int optind, opterr, optopt;
```

El parámetro **`optstring`** es una lista de caracteres

- cada uno representa una opción de un único caracter
- si está seguido de dos punto (:), indica que la opción tiene un valor asociado

El valor de regreso de **`getopt`** es el caracter de opción siguiente encontrado en el vector **`argv`** (si este existe).

Ejemplo:

```c
getopt(argc, argv, "if:lr")
```

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

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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

Archivos de dispositivos

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`
- Los dispositivos se clasifican en: dispositivos de caracteres o de bloques

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`
- Los dispositivos se clasifican en: dispositivos de caracteres o de bloques

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

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`
- Los dispositivos se clasifican en: dispositivos de caracteres o de bloques

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

- `open()`: Abrir archivo o dispositivo

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`
- Los dispositivos se clasifican en: dispositivos de caracteres o de bloques

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

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

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`
- Los dispositivos se clasifican en: dispositivos de caracteres o de bloques

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

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

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`
- Los dispositivos se clasifican en: dispositivos de caracteres o de bloques

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

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

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`
- Los dispositivos se clasifican en: dispositivos de caracteres o de bloques

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

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

Programación en lenguaje C en Linux

Manejo de archivos – funciones de bajo nivel

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 Hw se representan por archivos (mapean). Por ejemplo, el usuario root puede montar un disco USB como un archivos
  `$ mount -t vfat /dev/sdb /mnt/usbdisk`
- Los dispositivos se clasifican en: dispositivos de caracteres o de bloques

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

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

- Cada programa en ejecución (proceso) tiene un número de descriptores de archivo asociado.
- Estos son números enteros pequeños que se pueden utilizar para acceder a estos archivos o dispositivos.
- Cuando un programa se ejecuta están abierto tres de estos descriptores:
  - 0: entrada estándar
  - 1: salida estándar
  - 2: error estándar

- Cada programa en ejecución (proceso) tiene un número de descriptores de archivo asociado.
- Estos son números enteros pequeños que se pueden utilizar para acceder a estos archivos o dispositivos.
- Cuando un programa se ejecuta están abierto tres de estos descriptores:
  - 0: entrada estándar
  - 1: salida estándar
  - 2: error estándar

Ver ejemplos

- `simple_write`

- Cada programa en ejecución (proceso) tiene un número de descriptores de archivo asociado.
- Estos son números enteros pequeños que se pueden utilizar para acceder a estos archivos o dispositivos.
- Cuando un programa se ejecuta están abierto tres de estos descriptores:
  - 0: entrada estándar
  - 1: salida estándar
  - 2: error estándar

Ver ejemplos

- `simple_write`
- `simple_read` (Ver `/proc/<PID>`)
  Probar: `$ ./simple_read < draft.txt`

Programación en lenguaje C en Linux

Manejo de archivos – acceso de bajo nivel

Abrir archivo

```c
#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.

Programación en lenguaje C en Linux

Manejo de archivos – acceso de bajo nivel

Abrir archivo

```c
#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

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

int close(int fildes);
```

- `fildes`: descriptor de archivo

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`: Cantindad de bytes a leer/escribir
- `cmd`: Acción a realizar sobre el archivo