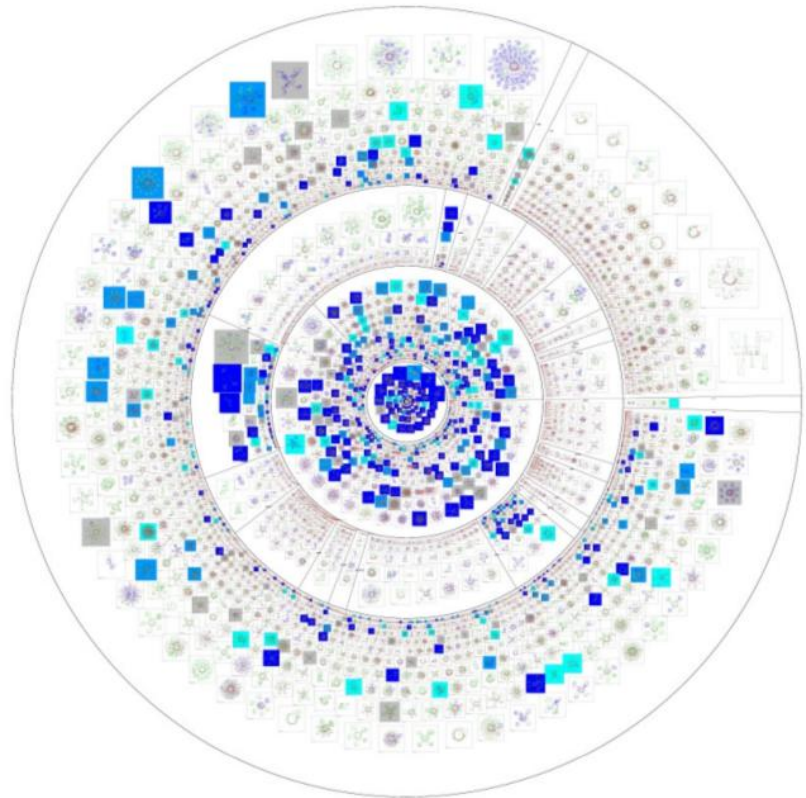
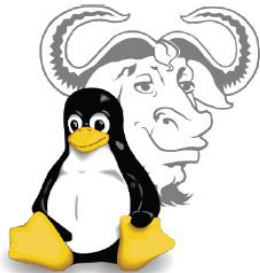


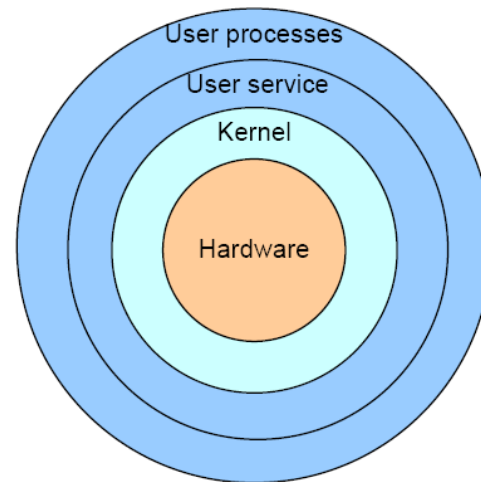
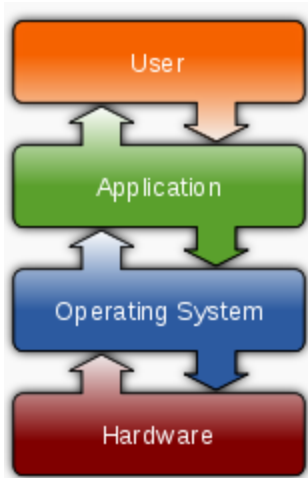
# User Space I/O

(CC) Por Daniel A. Jacoby



# Acceso al hardware en Linux

- Un Sistema operativo esta organizado en capas. Cada capa tiene una función especifica. Mediante este esquema se puede integrar fácilmente nuevo hardware

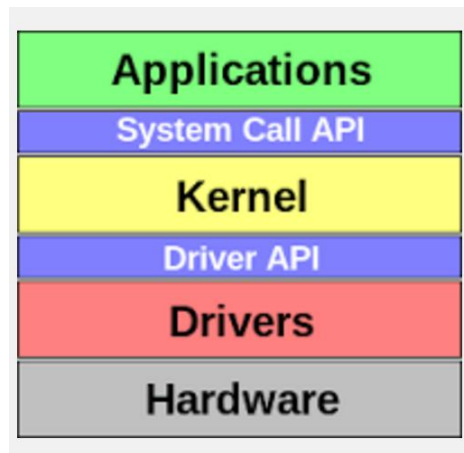


# Kernel y User space

- **Kernel Space** : Acceso directo al hardware de manera organizada. Impedir que el usuario acceda a recursos del hardware de cualquier forma
- **User space**: Aplicaciones del usuario que deberán estar controladas para evitar hacer daño al Sistema operativo u otras aplicaciones (Ring3)

# System Calls

- El acceso a las prestaciones del kernel en linux (y otros SO) se hace mediante el uso de una API (Application program interface) mediante un conjunto de funciones llamadas system calls.



## System Call Quick Reference

No	Func Name	Description
1	<a href="#"><u>exit</u></a>	terminate the current process
2	<a href="#"><u>fork</u></a>	create a child process
3	<a href="#"><u>read</u></a>	read from a file descriptor
4	<a href="#"><u>write</u></a>	write to a file descriptor
5	<a href="#"><u>open</u></a>	open a file or device
6	<a href="#"><u>close</u></a>	close a file descriptor
7	<a href="#"><u>waitpid</u></a>	wait for process termination

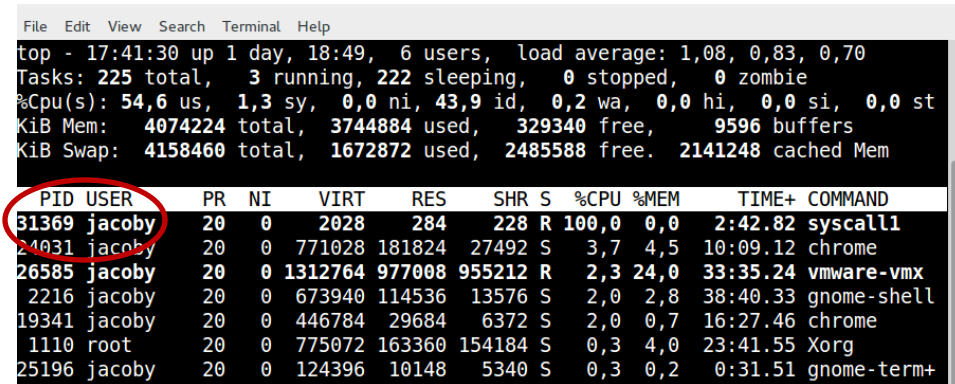
# SysCall Example 1: getpid

```
#include <syscall.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>

int main(void) {
    long ID1, ID2;
    /*-----*/
    /* direct system call          */
    /* SYS_getpid (func no. is 20) */
    /*-----*/
    ID1 = syscall(SYS_getpid);
    printf ("syscall(SYS_getpid)=%ld\n", ID1);
    /*-----*/
    /* "libc" wrapped system call */
    /* SYS_getpid (Func No. is 20) */
    /*-----*/
    ID2 = getpid();
    printf ("getpid()=%ld\n", ID2);
    while(1);
    return(0);
}
```

```
./syscall1
syscall(SYS_getpid)=31369
getpid()=31369
```

> top (show all processes)



File Edit View Search Terminal Help

top - 17:41:30 up 1 day, 18:49, 6 users, load average: 1,08, 0,83, 0,70  
Tasks: 225 total, 3 running, 222 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 54,6 us, 1,3 sy, 0,0 ni, 43,9 id, 0,2 wa, 0,0 hi, 0,0 si, 0,0 st  
KiB Mem: 4074224 total, 3744884 used, 329340 free, 9596 buffers  
KiB Swap: 4158460 total, 1672872 used, 2485588 free. 2141248 cached Mem

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
31369	jacoby	20	0	2028	284	228	R	100,0	0,0	2:42.82	syscall1
24031	jacoby	20	0	771028	181824	27492	S	3,7	4,5	10:09.12	chrome
26585	jacoby	20	0	1312764	977008	955212	R	2,3	24,0	33:35.24	vmware-vmx
2216	jacoby	20	0	673940	114536	13576	S	2,0	2,8	38:40.33	gnome-shell
19341	jacoby	20	0	446784	29684	6372	S	2,0	0,7	16:27.46	chrome
1110	root	20	0	775072	163360	154184	S	0,3	4,0	23:41.55	Xorg
25196	jacoby	20	0	124396	10148	5340	S	0,3	0,2	0:31.51	gnome-term+

# SysCall Example 2: write

```
#include <syscall.h>
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>

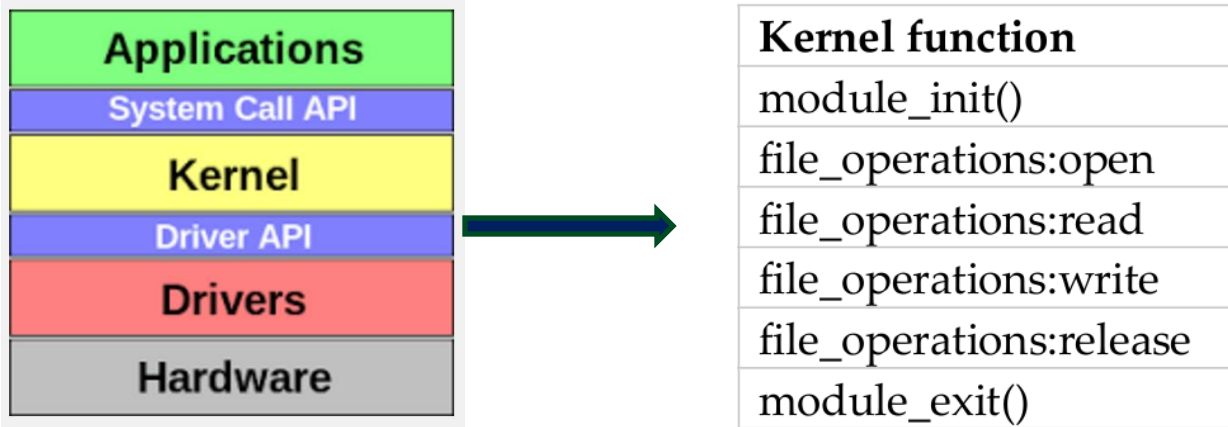
const char msg0[]="Hola ITBA (Direct System Call)\n";
const char msg1[]="Hola ITBA (Wraped System Call)\n";

int main(void) {
    long Nwritten;
    /*-----*/
    /* direct system call */
    /* SYS_write (func no. is 4) */
    /*-----*/
    Nwritten = syscall(SYS_write,STDOUT_FILENO,msg0,sizeof(msg0)-1);
    printf ("Number of bytes written=%ld\n", Nwritten);
    /*-----*/
    /* "libc" wrapped system call */
    /* SYS_write (Func No. is 4) */
    /*-----*/
    Nwritten = write(STDOUT_FILENO,msg1,sizeof(msg1)-1);
    printf ("Number of bytes written=%ld\n", Nwritten);
    return(0);
}
```

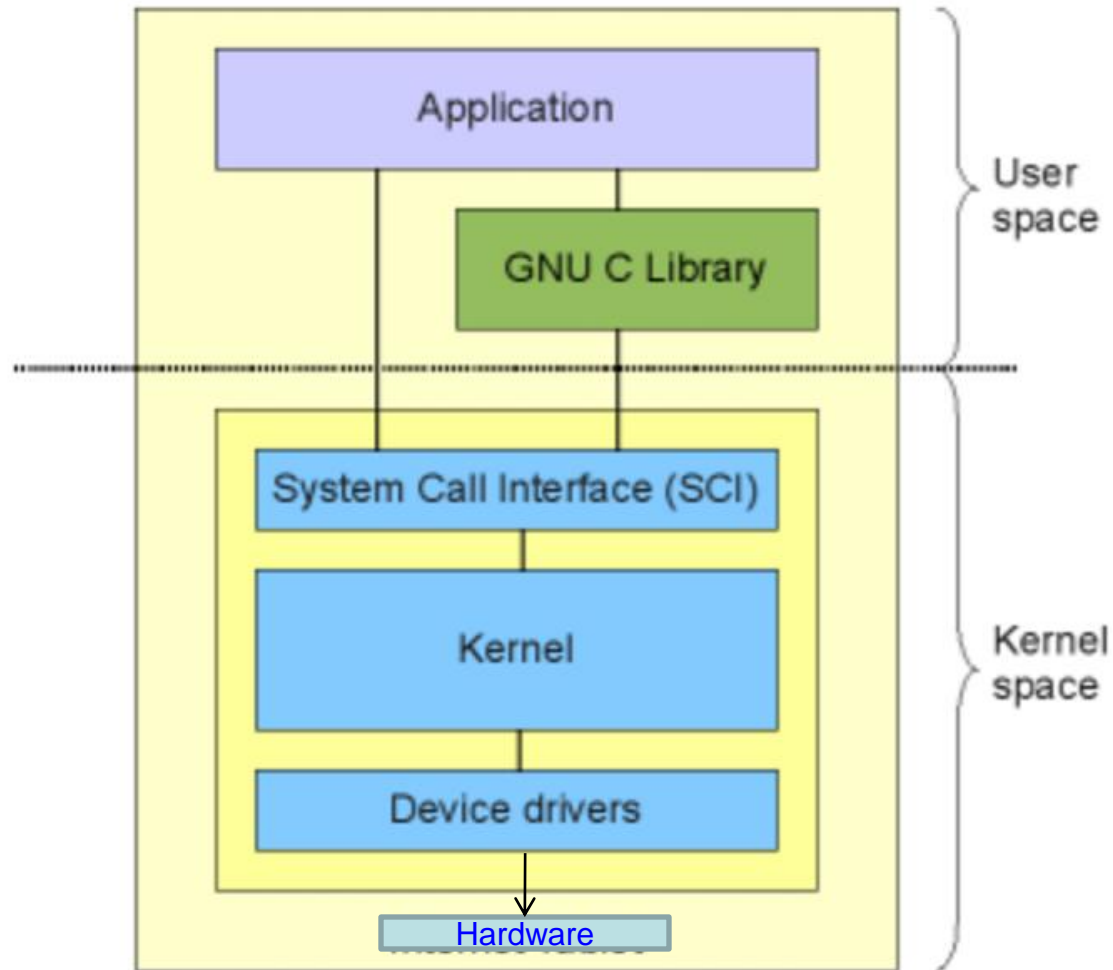
```
Hola ITBA (Direct System Call)
Number of bytes written=31
Hola ITBA (Wraped System Call)
Number of bytes written=31
```

# Device Drivers

- El acceso al hardware en linux (y otros SO) se hace mediante unos programas llamados Device Drivers.
- Los DD permiten que el Kernel interactúe con el hardware mediante el uso de una API (Driver API).



# Resumiendo

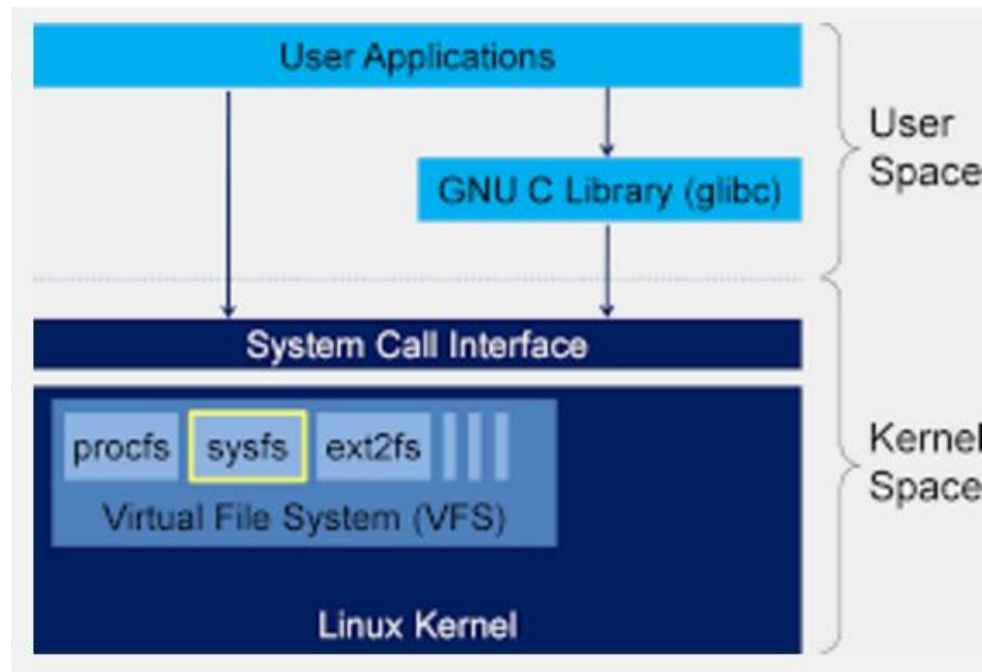




# SYSFS

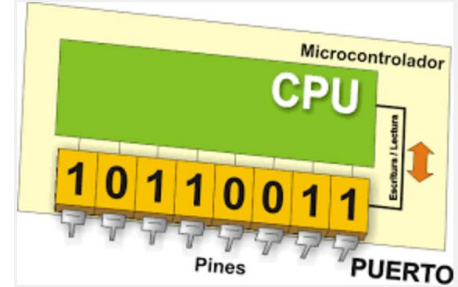
## Sysfs

- Es un sistema virtual de archivos que provee el kernel.
- Permite exportar información sobre dispositivos y drivers del kernel al user space.





# SYSFS-GPIO



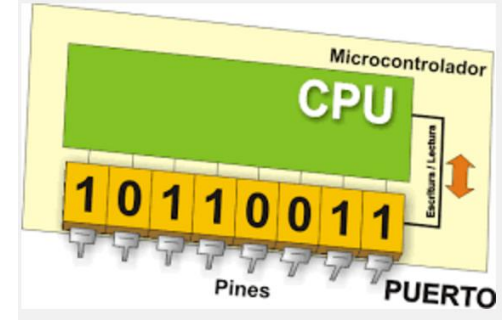
Sysfs esta montado en /sys

Estructura de Sysfs:

```
pi@raspberrypi ~ $ tree /sys/ -L 1
/sys/
├── block
├── bus
├── class
├── dev
├── devices
├── firmware
├── fs
├── kernel
├── module
└── power
```

```
pi@raspberrypi ~ $ tree /sys/class/ -L 1
/sys/class/
├── bdi
├── block
├── firmware
├── gpio
├── graphics
├── i2c-adapter
├── input
├── leds
├── mem
├── misc
├── mmc_host
├── net
├── raw
├── rtc
├── scsi_device
├── scsi_disk
├── scsi_host
├── sound
├── spi_master
├── thermal
├── tty
├── usb_device
├── vc
├── vchiq
├── vc-mem
└── vtconsole
```

# SYSFS-GPIO

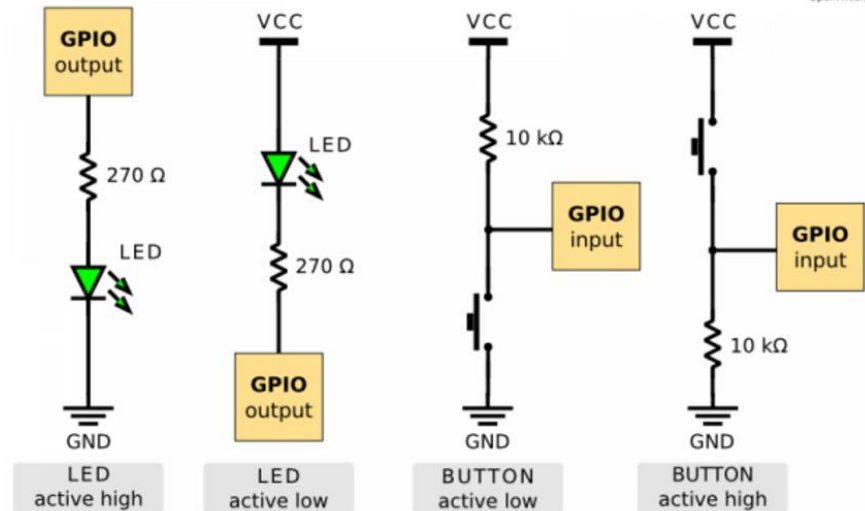
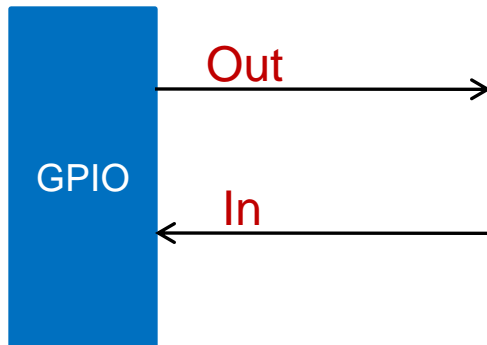
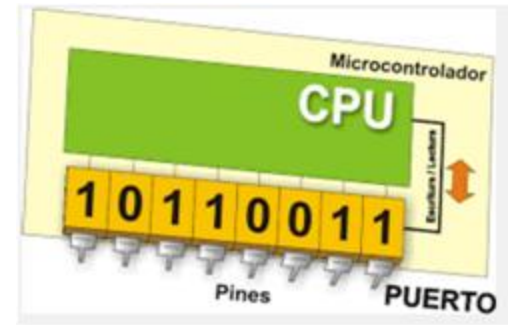


Estructura de GPIO:  
(Exported GPIO'S)

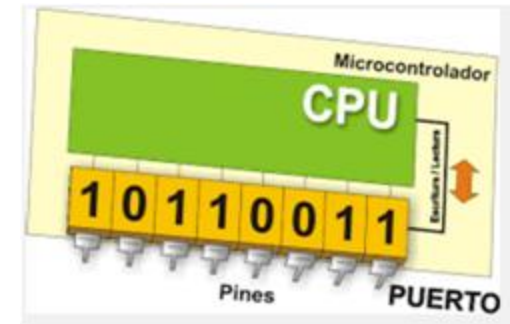
```
pi@raspberrypi ~ $ tree /sys/class/gpio/ -L 1
/sys/class/gpio/
├── export
├── gpio17 -> ../../devices/virtual/gpio/gpio17
├── gpio18 -> ../../devices/virtual/gpio/gpio18
├── gpio22 -> ../../devices/virtual/gpio/gpio22
├── gpio23 -> ../../devices/virtual/gpio/gpio23
├── gpio24 -> ../../devices/virtual/gpio/gpio24
├── gpio25 -> ../../devices/virtual/gpio/gpio25
├── gpio27 -> ../../devices/virtual/gpio/gpio27
├── gpio4 -> ../../devices/virtual/gpio/gpio4
├── gpiochip0 -> ../../devices/virtual/gpio/gpiochip0
└── unexport
```

# GPIO

GPIO: General Purpose Input Output



# GPIO



Uso desde la linea de comandos (CLI) como super usuario

**echo "17" > /sys/class/gpio/export** // Exportamos pin 17 al user space

Despues de exportar en pin 17 aparecen los siguientes directorios en user space

```
root@raspberrypi:/home/pi/Simon# tree /sys/class/gpio/gpio17
/sys/class/gpio/gpio17
├── active_low
├── direction
├── edge
├── power
├── subsystem -> ../../../../class/gpio
├── uevent
└── value
```

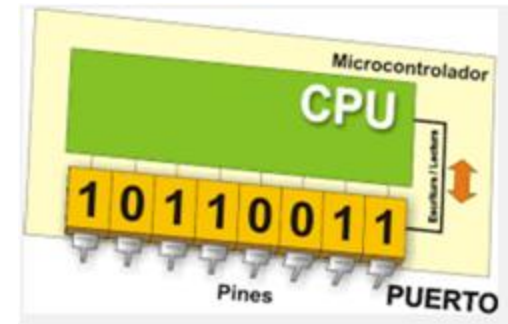
**active\_low** : controla la polaridad del pin: Write 0 → normal Write 1 → invierte

**direction**: controla el sentido: Write "in" → Entrada Write "out" → Salida

**value**: Si el pin es salida : Write "0" → Pone el Pin en 0 Write "1" Pone el Pin en 1  
Si el pin es entrada devuelve el estado del pin al leer.

**Nota:** As non root user : echo "17" | sudo tee /sys/class/gpio/export

# GPIO



Uso desde la linea de comandos (CLI)

Ejemplo Completo (Salida):

```
echo "17" > /sys/class/gpio/export // Exportamos pin 17 al user space
```

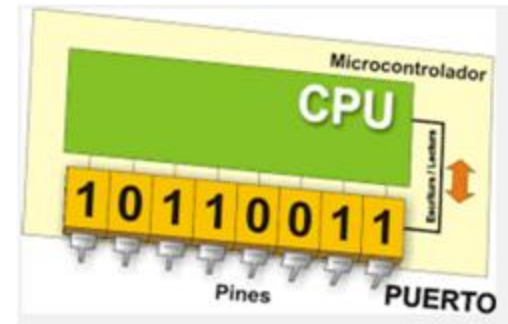
```
echo "out" > /sys/class/gpio/gpio17/direction // Fijamos pin 17 como salida
```

```
echo "1" > /sys/class/gpio/gpio17/value // Pin 17 en '1'
```

```
echo "0" > /sys/class/gpio/gpio17/value // Pin 17 en '0'
```

```
echo "17" > /sys/class/gpio/unexport // Devolvemos pin 17 al kernel space
```

# GPIO



Uso desde la linea de comandos (CLI)

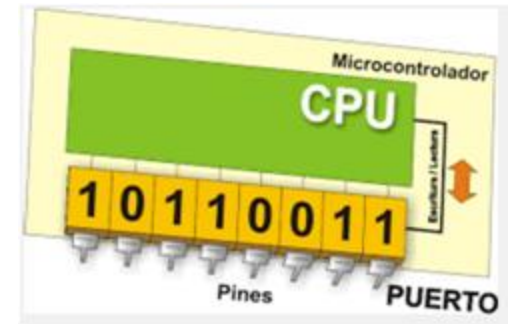
Ejemplo Completo(Entrada):

```
echo "22" > /sys/class/gpio/export // Exportamos pin 22 al user space
```

```
echo "in" > /sys/class/gpio/gpio22/direction // Fijamos pin 22 como salida
```

```
cat /sys/class/gpio/gpio22/value // Leemos Pin 22
```

# GPIO



Uso desde la linea de comandos (CLI)

Podemos verificar los valores actuales

Para ver que exportamos: **ls /sys/class/gpio/**

```
root@raspberrypi:/home/pi# ls /sys/class/gpio/  
export gpio17 gpio18 gpio22 gpio23 gpio24 gpio25 gpio27 gpio4 gpiochip0 unexport
```

Para ver la direccion de un pin : **cat /sys/class/gpio/gpio25/direction**

```
root@raspberrypi:/home/pi# cat /sys/class/gpio/gpio25/direction  
out
```

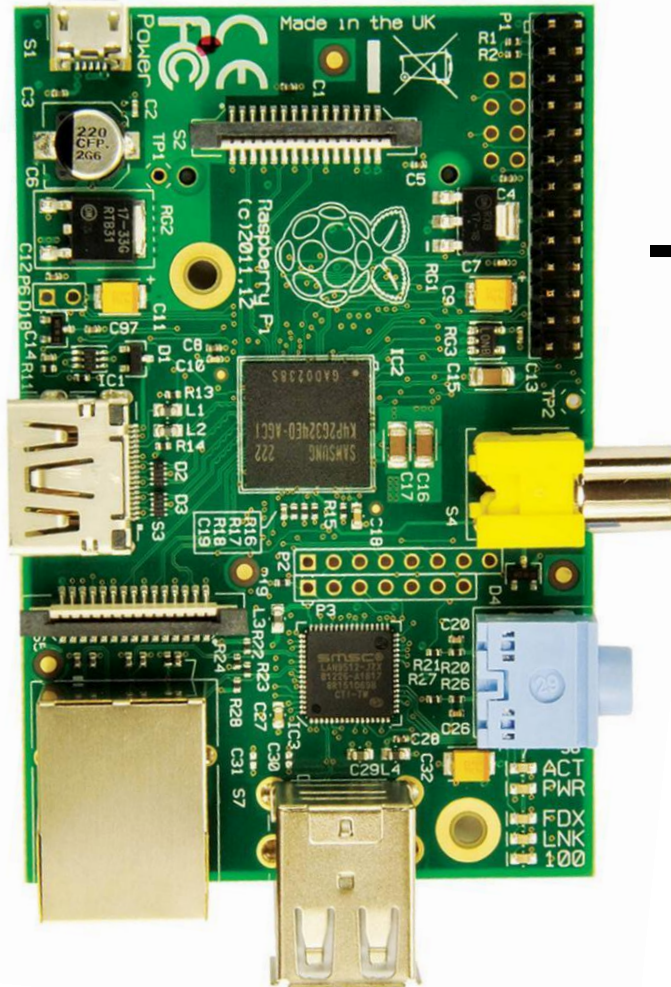
Para ver el valor de un pin : **cat /sys/class/gpio/gpio27/value**

```
root@raspberrypi:/home/pi# cat /sys/class/gpio/gpio27/value  
1
```

Esto ultimo es valido ya sea el pin entrada o salida

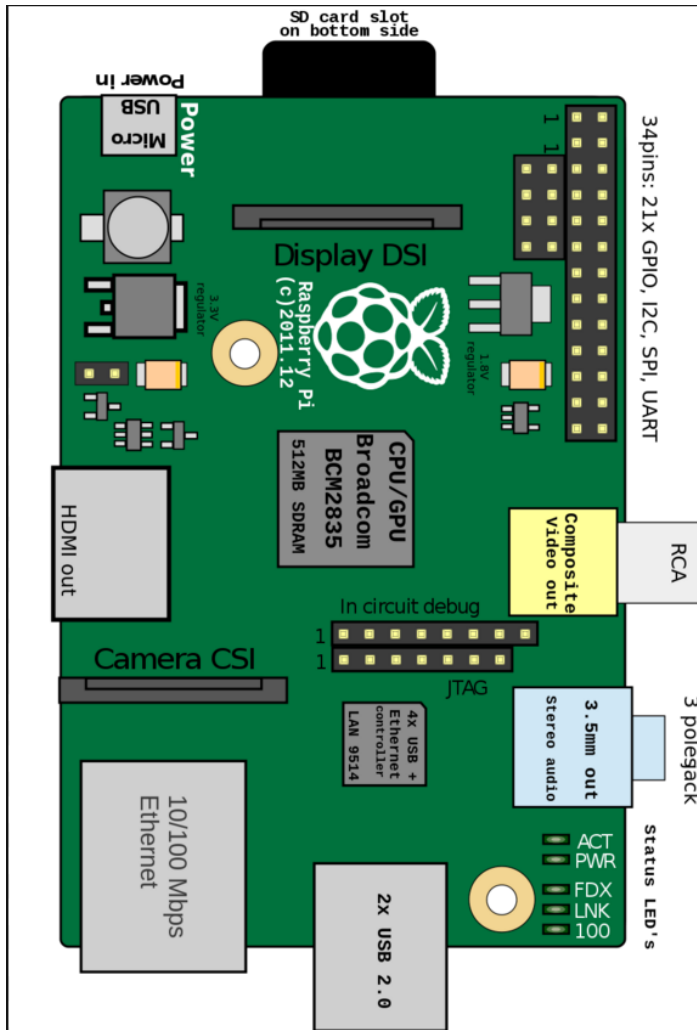


# Raspberry PI Modelo B



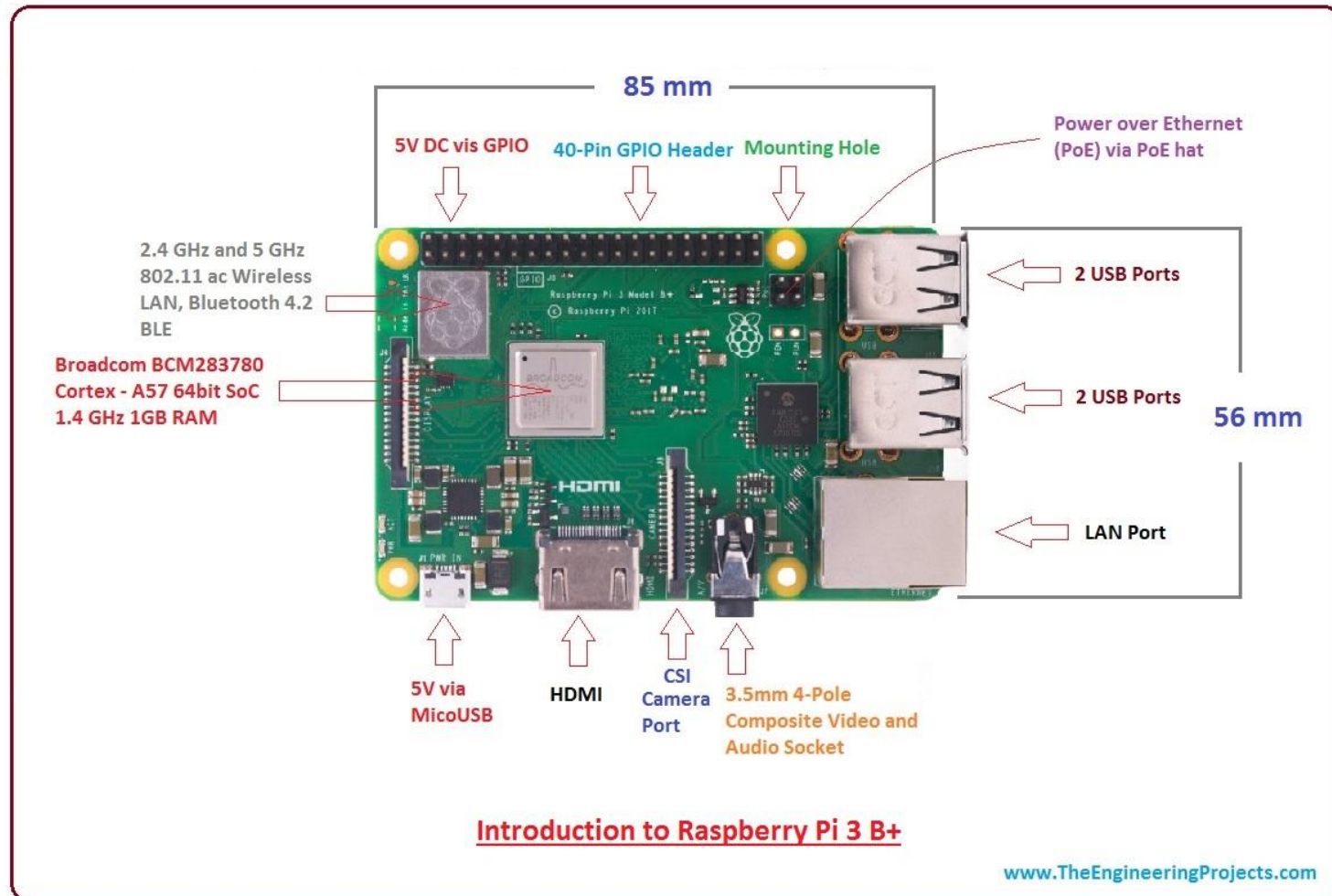
Pin 1	Pin 2		
+3V3			+5V
GPIO2 / SDA1			+5V
GPIO3 / SCL1			GND
GPIO4			TXD0 / GPIO 14
GND			RXD0 / GPIO 15
GPIO17			GPIO 18
GPIO27			GND
GPIO22			GPIO 23
+3V3			GPIO 24
GPIO10 / MOSI			GND
GPIO9 / MISO			GPIO 25
GPIO11 / SCLK			CE0# / GPIO8
GND			CE1# / GPIO7
Pin 25	Pin 26		

# Raspberry Pi Modelo B



Pin 1	Pin 2
+3V3	+5V
GPIO2 / SDA1	+5V
GPIO3 / SCL1	GND
GPIO4	TXD0 / GPIO 14
GND	RXD0 / GPIO 15
GPIO17	GPIO 18
GPIO27	GND
GPIO22	GPIO 23
+3V3	GPIO 24
GPIO10 / MOSI	GND
GPIO9 / MISO	GPIO 25
GPIO11 / SCLK	CE0# / GPIO8
GND	CE1# / GPIO7
Pin 25	Pin 26

# Raspberry Pi 3 B+



# Raspberry Pi 3 B+



Pin 1	Pin 2
+3V3	+5V
GPIO2 / SDA1	+5V
GPIO3 / SCL1	GND
GPIO4	TXD0 / GPIO 14
GND	RXD0 / GPIO 15
GPIO17	GPIO 18
GPIO27	GND
GPIO22	GPIO 23
+3V3	GPIO 24
GPIO10 / MOSI	GND
GPIO9 / MISO	GPIO 25
GPIO11 / SCLK	CE0# / GPIO8
GND	CE1# / GPIO7
Pin 25	Pin 26

# Programación de GPIO usando la API

Dado que el kernel exporta los pines como archivos podemos usar:

1- Las funciones que nos brinda glibc (la librería de c) para acceder a archivos.

Ej : `fopen()` `fwrite()`, etc.

2- Los wrappers de los System Calls del kernel que nos ofrece glibc (la librería de c).

Ej: `open()`, `write()`, etc.

# Programación de GPIO usando la API

`fopen()` vs. `open()` ?





# Programación de GPIO usando la API

## fopen() vs. open()

fopen() (High Level)	Open() (Low level)
Es una función de la librería de c	Es un wrapper de un System Call (5)
Buffered I/O	Unbuffered I/O
Mas Rapida (No switch to KS all the time)	Mas lenta (switch from US to KS always)
Es portable	No es portable
Usa un file pointer FILE * fp=fopen(...)	Usa un file descriptor int fd =open(.....)

En nuestro caso es indistinto cual usar pues escribimos un carácter por vez para cambiar el estado del pin. La unica diferencia es que debemos llamar a **fflush()** cada vez que escribimos al pin si usamos fopen() o bien hacer un **fclose()** si no necesitamos dejar abierto el archivo.

# Programación de GPIO usando la API

## Ejemplos: Export pin 22

```
FILE *handle_export;
int nWritten;

if ((handle_export = fopen("/sys/class/gpio/export", "w")) == NULL)
{
    printf("Cannot open EXPORT File. Try again later.\n");
    exit(1);
}
nWritten=fputs("22",handle_export);
if (nWritten==-1)
{
    printf ("Cannot EXPORT PIN . Try again later.\n");
    exit(1);
}
else
    printf("EXPORT File opened succesfully \n");

fclose(handle_export); // Be carefull do this for EACH pin !!!
```



# Programación de GPIO usando la API

## Ejemplos: Set pin 22 as output

```
FILE * handle_direction;
int nWritten;
if ((handle_direction = fopen("/sys/class/gpio/gpio22/direction", "w")) == NULL)
{
    printf("Cannot open DIRECTION File");
    exit(1);
}
// Set pin Direction

if ((nWritten=fputs("out",handle_direction))== -1)
{
    printf("Cannot open DIRECTION pin. Try again later.\n");
    exit(1);
}
else
{
    printf("DIRECTION File for PIN opened succesfully\n");
}

fclose(handle_direction); // Be carefull do this for EACH pin !!!
```

# Programación de GPIO usando la API

## Ejemplos: Set pin 22 Low

```
FILE * handle;
int nWritten;
char *pin22 = "/sys/class/gpio/gpio22/value";
void main(void)
{
    if ((handle = fopen(pin22,"w")) == NULL)
    {
        printf("Cannot open device. Try again later.\n");
        exit(1);
    }
    else
    {
        printf("Device successfully opened\n");
    }

    if(fputc('0' ,handle)==-1) // Set pin low
    {
        printf("Clr_Pin: Cannot write to file. Try again later.\n");
        exit(1);
    }
    else
        printf("Write to file %s successfully done.\n",pin22);

    fclose(handle);
}
```

# Uso de la Raspberry PI

Podemos conectarnos a la placa de 2 formas:

- Por WiFi
- Por cable Ethernet

En ambos casos no será necesario conocer el IP asignado a la placa.

La forma de ingresar a la placa es la misma usando el protocolo SSH

En Ubuntu abrimos una terminal (ctrl-alt-T) y ingresamos:

- ssh [pi@raspberrypi.local](https://pi.raspberrypi.local)
- pi@raspberrypi.local's password: **gedageda**

# Uso de la Raspberry PI

Si todo funciona bien veremos lo siguiente:

```
jacoby@jacoby-ThinkPad-P50:~$ ssh pi@raspberrypi.local
pi@raspberrypi.local's password:
Linux raspberrypi 4.14.98-v7+ #1200 SMP Tue Feb 12 20:27:48 GMT 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
No mail.
Last login: Mon Jun  3 11:24:20 2019 from 10.42.0.1
pi@raspberrypi:~ $ |
```

# Conectarse vía WiFi

## Setup



# Conectarse vía WiFi

1- Usando el teléfono celular crear un Hotspot

Los parámetros deberán ser:

**Nombre de la red:** GrupoX X= Numero de grupo

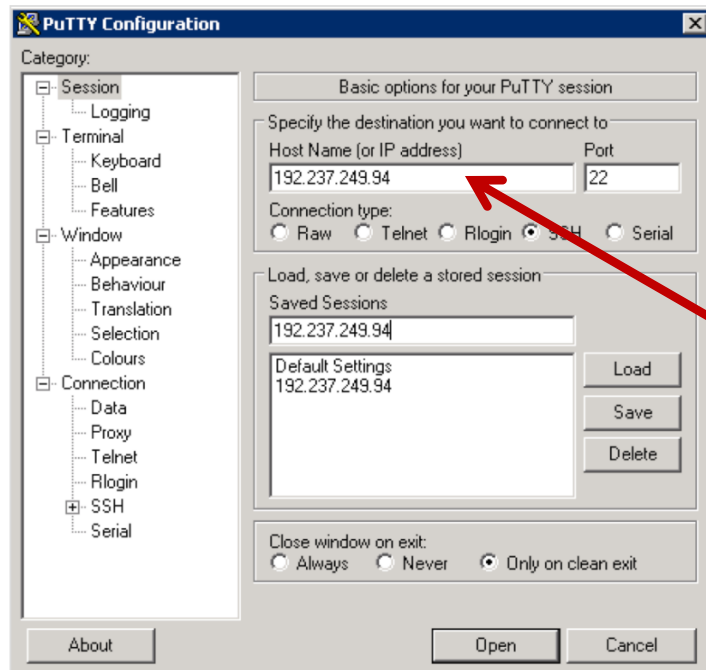
**Password:** gedageda

La conexión debería establecerse en menos de 30 seg una vez activado el hotspot .

2- Conectarse con la PC a la red recién creada

# Usuarios de Windows

1- Instalar como cliente de SSH PuTTY <https://www.putty.org/>



[pi@raspberrypi.local](#)

Nota: si no funciona instalar Bonjour para windows [Bonjour](#)  
[https://support.apple.com/kb/DL999?viewlocale=en\\_US&locale=en\\_US](https://support.apple.com/kb/DL999?viewlocale=en_US&locale=en_US)

# Usuarios de Windows

**Nota:** Si no funciona instalar Bonjour para windows [Bonjour](#)

[https://support.apple.com/kb/DL999?viewlocale=en\\_US&locale=en\\_US](https://support.apple.com/kb/DL999?viewlocale=en_US&locale=en_US)



Download Bonjour Print Services for Windows  
v2.0.2

Download



# Uso de la Raspberry PI

Otra alternativa es conectar la placa directamente a la computadora usando un cable de red entre la dos.

En Ubuntu:



Nota : En Ubuntu se puede abrir el editor de conexiones de red el siguiente comando desde consola : **nm-connection-editor**

# Uso de la Raspberry PI

Para copiar archivos desde nuestra PC a la raspberry pi usamos el comando **scp**

1- Crear EN la RPI una carpeta Ej: **mkdir Test**

2- Copiar un archhivo desde nuestra PC a la RPI

```
scp ejemplo_pin.c pi@raspberrypi.local:/home/pi/Test
```

Para copiar carpetas

```
scp -r SysCalls/ pi@raspberrypi.local:/home/pi/Test
```

Copia la carpeta Syscalls desde nuestra PC a la carpeta Test en la RPI

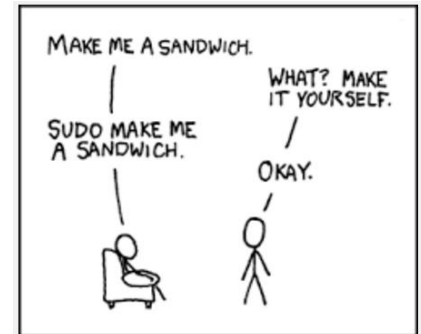
Para compilar en la RPI usar gcc en la forma habitual

# Uso de la Raspberry PI

Para ejecutar programas que hacen uso de recursos del hardware se requiere ser **super usuario** (administrador)

Esto se puede hacer de 2 formas

- 1- Convertirse en SU temporariamente: **sudo ./mi\_ejecutable**
- 2- Convertirse en SU en forma permanente (No recomendado)
  - a - **sudo su**
  - b - **./mi\_ejecutable**



Usuario pi → 

```
pi@raspberrypi ~/Simon $
```

Usuario root → 

```
pi@raspberrypi ~/Simon $
```

```
pi@raspberrypi ~/Simon $ sudo su
```

```
root@raspberrypi:/home/pi/Simon#
```

# Uso de la Raspberry PI

## Power Down

Rogamos que apaguen correctamente la placa después de usarla

De no hacerlo así se puede corromper el sistema de archivos



**No la desconecten de la alimentación !!!!**

**Primero:** ejecutar desde la línea de comandos lo siguiente :

**sudo shutdown -h now**

**Segundo:** Esperar unos 30 segundos

(El led Verde muestra actividad apenas se ejecuta el comando luego se detiene esta actividad y después de unos 15 segundos vuelve a haber actividad finalizando el proceso)

Recién entonces desconectar la fuente!!!!

ANY  
QUESTIONS  
?