UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DE COMPUTADORES

# DRIVER LINUX PARA PANTALLA OLED SSD1306 I2C

# LINUX DRIVER FOR SSD1306 I2C OLED DISPLAY


Realizado por
**José Carlos Navarro González**
Tutorizado por
**Andrés Rodríguez Moreno**
Departamento
**Arquitectura de Computadores**

UNIVERSIDAD DE MÁLAGA


MÁLAGA, JUNIO DE 2018

Fecha defensa:

El secretario del Tribunal:

## RESUMEN

Este proyecto consiste en la realización de un driver de dispositivos para Linux para poder usar fácilmente desde el espacio de usuario una pequeña pantalla Oled SSD1306 conectada al bus serie i2c, para ello hemos utilizado la placa de desarrollo Raspberry Pi, ya que a pesar de su bajo coste nos permite correr un sistema operativo Linux y nos ofrece una serie de GPIOs (E/S de propósito general) para conectar dispositivos, entre los cuales se encuentra en dos de ellos el bus i2c.

El driver crea un cliente i2c en un módulo cargable en el kernel de Linux para la pantalla oled que permitirá mostrar texto haciendo scrolll automáticamente y el borrado de esta.

Otra parte del trabajo ha sido añadir mediante un Device Tree Overlay la descripción del nuevo dispositivo a incorporar al sistema, la realización de una librería C para poder usar la pantalla desde un lenguaje de alto nivel y un par de servicios de Linux (Systemd), uno para instalar el driver en el arranque del sistema y otro que tras el arranque muestre en la pantalla Oled la dirección IP de la placa.

El objetivo final de este trabajo es incorporar el proyecto y la documentación al material de apoyo de la asignatura de Diseño de Sistemas Operativos del Grado de Ingeniería de Computadores.



## PALABRAS CLAVE

Raspberry Pi, controlador de dispositivo, Linux, Device Tree, Systemd, i2c, GPIO, sistema operativo, oled.

## ABSTRACT

This project consists in the creation of a device driver for Linux, to use easily a small Oled SSD1306 screen connected to the i2c serial bus from the user space. To do this, it has been used the Raspberry Pi development board, because despite its low cost, we can run a Linux operating system and it provides us a series of GPIOs (E/S for general purpose) to connect some devices, among which are the i2c bus in two of them.

The driver creates an i2c client in a module and it can be charged in the Linux kernel to the Oled screen that allows to show text doing automatically scroll and the screen erasing.

Another part of the work has been to add the description of the new device that I have to incorporate to the system, all of this it was made by a Device Tree Overlay; the production of a C library to use the screen from a high level language and a pair of Linux's services (Systemd); one to install the driver in the system start up; and other that shows the board IP address in the Oled screen after the start.

The final goal of this work is to incorporate the project and the documentation to the supporting material of the Design of Operating Systems' subject in the Computer Engineering Degree.

## KEYWORDS

Raspberry Pi, device driver, Linux, Device Tree, Systemd, i2c, GPIO, operating system, oled

# ÍNDICE

# 1.- INTRODUCCIÓN

## 1.1.- Objetivo.

El objetivo de este proyecto ha sido el estudio de los controladores de dispositivos, más especialmente los de caracteres, para ello se ha realizado un ejemplo de controlador para una pantalla Oled y se ha intentado reflejar todo en esta memoria para que pueda servir de referencia para cualquier otra persona que desee realizar un controlador para un dispositivo similar.

Por lo dicho anteriormente, se va a poder añadir al material para la asignatura de Diseño de Sistemas Operativos, en la cual se dan las nociones básicas para realización de controladores de dispositivos para el kernel de Linux.

## 1.2.- Contenido y estructura de la memoria.

La memoria se encuentra dividida en 8 apartados principales:

- Un primer apartado de introducción con el cuál se intenta hacer una idea general del proyecto, para que cualquier persona que lea este apartado decida si está interesado en aprender más sobre él o no.

- En el segundo apartado se explica en que consiste el bus i2c, así como un poco de historia de este y su funcionamiento, tanto en general, como especialmente en Raspberry pi, ya que ha sido el utilizado en este proyecto.

- En el tercer apartado, se define que es un controlador de dispositivos basándonos siempre en Linux, y la explicación de los dos tipos de controladores de dispositivos más usuales, también se verá la diferencia entre el espacio de usuario y el espacio del kernel.

- En el cuarto apartado, se hará una breve introducción a los servicios de systemd, ya que se utilizarán para cargar el módulo del dispositivo y algún otro script en el arranque del sistema.

- El quinto aparado habla sobre el Device Tree, algunos conceptos necesarios para desarrollar éste y para su compilación. Y se explicará en que consiste el Device Tree Overlay.

- En el sexto apartado se explica la realización del proyecto completo, donde se definen todas las funciones que se han utilizado tanto para el driver como para Device Tree, librería C para la pantalla, script para ip y servicios de systemd.

- En el séptimo apartado se explican los pasos uno a uno para instalar el driver y añadirlo al arranque del sistema, si se realizan todos los pasos, debería funcionar la pantalla.

- Por último, el octavo apartado trata las conclusiones del proyecto, y se analizan algunas mejoras que podrían aplicarse para líneas futuras.

# 2.- BUS I2C

## 2.1.- Introducción.

El bus i2c (inter integrated circuits) es un estándar basado en un bus maestro-esclavo con una metodología de comunicación de datos en serie y síncrona, introducido por Philips en 1982, que facilita la comunicación interna entre circuitos integrados y otros dispositivos. Desde esa fecha hasta la actualidad han ido surgiendo una serie de versiones, que se describen brevemente en el siguiente apartado, las cuales han ido mejorando el bus.

## 2.2.- Versiones.

- La primera versión de especificación estandarizada se publicó en 1992 y se le llamo modo rápido (FM) la cual sustituyó el estándar original de 100 kbps por uno más rápido de 400 kbps y aumentó el espacio de direccionamiento a un modo de 10 bits.

- En 1998 se publicó la segunda versión, con el modo de alta velocidad (Hs), llegando a un máximo de 3,4 Mbps, pero con la consecuencia de que los requisitos de intensidad y voltaje fueron reducidos.

- Con la tercera versión, publicada en 2007 se incluyó un modo nuevo denominado Fm+ que mejoraría el modo rápido llegando a una velocidad de 1 Mbps y volviendo a utilizar el mismo protocolo que los modos de 100 y 400 Kbps.

- La cuarta versión llegó en 2012, a la cual se le llamo Ufm (modo ultrarrápido) que es compatible con velocidades de transferencias unidireccionales de hasta 5 Mbps.

- Entre finales de 2012 y 2015 fueron presentadas las versiones 5 y 6 respectivamente, que se basaron en la corrección de algunos errores que habían surgido en la versión 4.

## 2.3.- ¿Qué es i2c?

El bus i2c, es un bus bidireccional que utiliza dos líneas de datos, una de datos serie (SDA) y otra línea de reloj serie (SCL) que se utiliza para sincronizar los datos de SDA durante las transferencias mediante el bus, el cual trabaja con lógica positiva, es decir, el nivel bajo en la línea de datos es un 0 lógico y el nivel alto un 1 lógico.

Estas líneas tienen un estado similar al de un colector abierto, pero asociadas a un transistor de efecto de campo (FET) y se deben polarizar en estado alto utilizando resistencias “pull-up” para conectarlas a la alimentación, lo que hace que permita conectar en paralelo múltiples entradas y salidas.

Como se ha comentado anteriormente el bus se basa en el sistema de comunicación maestro-esclavo, por lo que la transferencia de datos siempre es iniciada por un maestro que es a la vez el que se encarga de manejar el reloj (línea SCL). También está la posibilidad de conectar varios maestros a la vez, pudiendo hacer todos ellos las funciones de esclavo cuando sea necesario.

Todas las direcciones del bus son de 7 o de 10 bits por lo que se pueden conectar hasta 128 dispositivos a él.

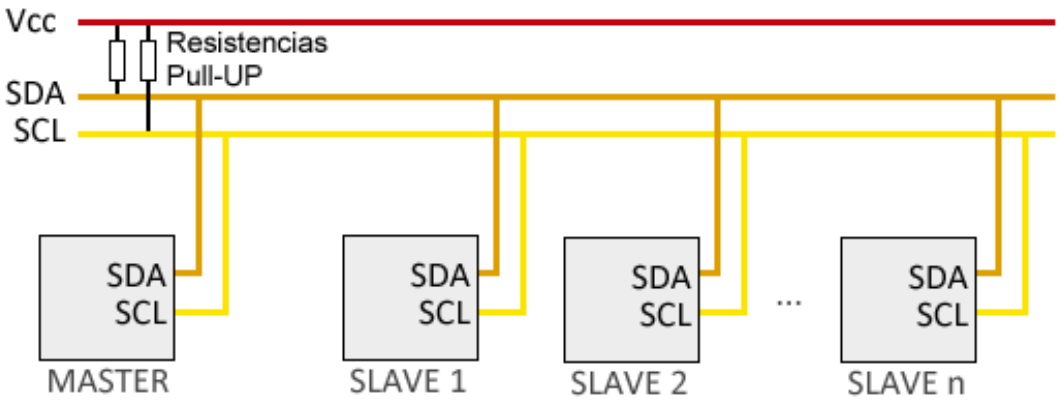



*Ilustración 1. ESQUEMA BUS I2C*

## 2.4.- Protocolo de comunicación del bus i2c.

Al poderse conectar varios dispositivos al bus, este debe respetar un protocolo que vamos a describir a continuación:

- En el bus tenemos conectado dos tipos de dispositivos, maestros y esclavos, solo los maestros pueden ocupar el bus.

- Cualquier dispositivo maestro puede ocupar el bus si este está libre, es decir, cuando ambas señales están en estado lógico alto, para ello establece la condición de start poniendo a nivel bajo su línea de datos (SDA) y dejando en nivel alto la línea de reloj. (SCL).

- Una vez establecida la condición de start, se trasmite el primer byte, cuyos siete primeros bits componen la dirección del dispositivo que se desea seleccionar, y el octavo y último bit la operación que se desea realizar con él (escritura o lectura).

- Si el dispositivo cuya dirección índica los 7 bits primeros del primer byte enviado por el maestro está presente en el bus, éste contesta con un bit en bajo (ACK), ubicado inmediatamente luego del octavo bit que ha enviado el maestro e indica que está listo para comunicarse y comenzar el intercambio de información entre los dispositivos.

## 2.5.- Lectura y escritura en i2c.

Cuando el bit que nos índica si desea realizar una lectura o una escritura ésta en baja, indica que el maestro va a escribir información en el dispositivo esclavo y para ello sigue los siguientes pasos:

1. Envía secuencia de inicio.
2. Envía la dirección del dispositivo con el bit de lectura/escritura en bajo.
3. Envía en número de registro interno en el que desea escribir.
4. Envía los bytes de datos.
5. Envía la secuencia de parada.

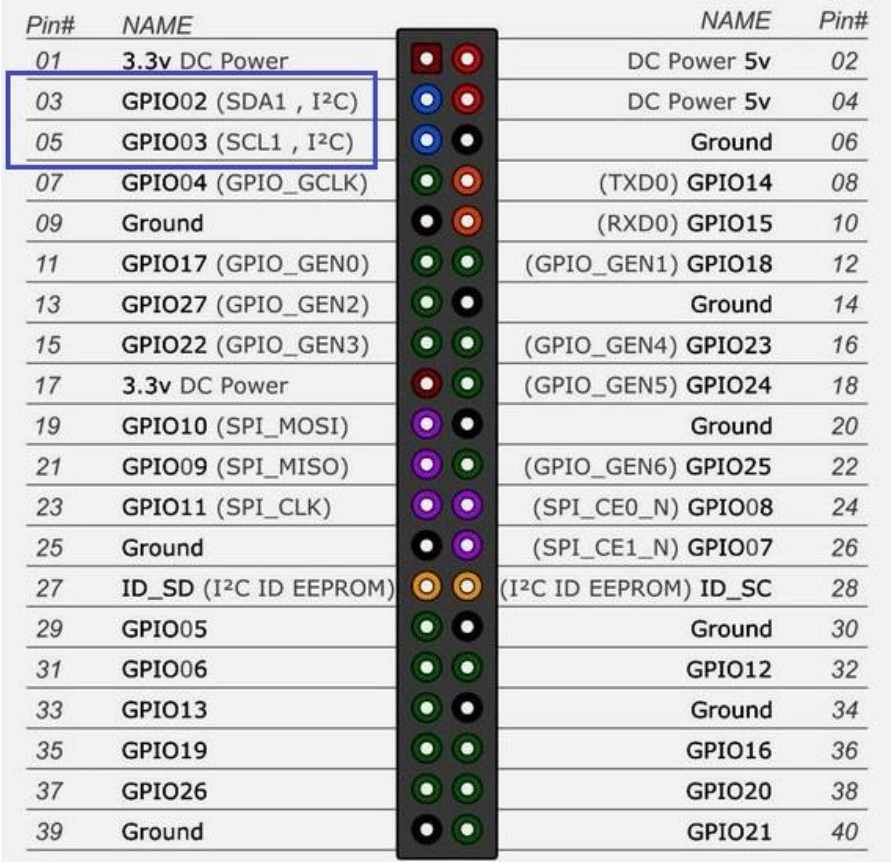
Para el caso de que el maestro quiera leer desde un dispositivo esclavo, la operación es un poco más complicada, se describe a continuación:

1. Envía secuencia de inicio.
2. Envía la dirección del dispositivo con el bit de lectura/escritura en bajo.
3. Envía dirección interna del registro del esclavo desde el que va a leer.
4. Envía de nuevo una secuencia de inicio.
5. Envía la dirección del dispositivo con el bit de lectura/escritura en alto.
6. Lee los bytes de datos.
7. Envía secuencia de parada.

## 2.6.- Bus i2c en Raspberry Pi.

La Raspberry Pi incorpora el modo estándar de 100 kbps y el modo rápido de 400 kbps y dispone de dos periféricos para implementar i2c, el BSC (Broadcom Serial Controller) que implementa el modo maestro y el BSI (Broadcom Serial Interface) que implementa el modo esclavo, nos vamos a centrar únicamente en el BSC ya que es el modo que se ha utilizado para realizar este proyecto.

El BSC implementa tres maestros independientes que tienen que estar en buses i2c separados ya que en este caso no se permite el multimaestro, el BSC0 se reserva para la identificación de las placas de expansión y el BSC2 es para la interfaz HDMI, por lo que solo queda libre para utilizar el BSC1 utilizando los pines 3 y 5 del conector J8 como se ve en la siguiente ilustración.



| Pin# | NAME | | NAME | Pin# |
|---|---|---|---|---|
| 01 | **3.3v** DC Power | | DC Power **5v** | 02 |
| 03 | **GPIO02** (SDA1 , I²C) | | DC Power **5v** | 04 |
| 05 | **GPIO03** (SCL1 , I²C) | | **Ground** | 06 |
| 07 | **GPIO04** (GPIO_GCLK) | | (TXD0) **GPIO14** | 08 |
| 09 | **Ground** | | (RXD0) **GPIO15** | 10 |
| 11 | **GPIO17** (GPIO_GEN0) | | (GPIO_GEN1) **GPIO18** | 12 |
| 13 | **GPIO27** (GPIO_GEN2) | | **Ground** | 14 |
| 15 | **GPIO22** (GPIO_GEN3) | | (GPIO_GEN4) **GPIO23** | 16 |
| 17 | **3.3v** DC Power | | (GPIO_GEN5) **GPIO24** | 18 |
| 19 | **GPIO10** (SPI_MOSI) | | **Ground** | 20 |
| 21 | **GPIO09** (SPI_MISO) | | (GPIO_GEN6) **GPIO25** | 22 |
| 23 | **GPIO11** (SPI_CLK) | | (SPI_CE0_N) **GPIO08** | 24 |
| 25 | **Ground** | | (SPI_CE1_N) **GPIO07** | 26 |
| 27 | **ID_SD** (I²C ID EEPROM) | | (I²C ID EEPROM) **ID_SC** | 28 |
| 29 | **GPIO05** | | **Ground** | 30 |
| 31 | **GPIO06** | | **GPIO12** | 32 |
| 33 | **GPIO13** | | **Ground** | 34 |
| 35 | **GPIO19** | | **GPIO16** | 36 |
| 37 | **GPIO26** | | **GPIO20** | 38 |
| 39 | **Ground** | | **GPIO21** | 40 |

*Ilustración 2. GPIOS Raspberry pi*

# 3.- CONTROLADOR DE DISPOSITIVOS

## 3.1.- Introducción.

Un controlador de dispositivos o también conocido su término en inglés “Device Driver”, es un programa software que tiene como objetivo controlar y administrar un dispositivo hardware. Desde el punto de vista del sistema operativo este puede estar en el espacio del kernel (se ejecuta en modo privilegiado) o en el espacio de usuario (se ejecuta con un privilegio menor), a continuación, se van a definir los dos espacios nombrados anteriormente.

## 3.2.- Espacios Kernel/Usuario.

- **Espacio del Kernel:** Es un rango de direcciones de memoria donde se aloja el kernel que está protegido por indicadores de acceso, lo que impide que se mezclen las aplicaciones de usuario con el kernel.

  El kernel se ejecuta con mayor prioridad por lo que puede acceder a toda la memoria del sistema (tanto al espacio del kernel como al espacio de usuario).

- **Espacio de Usuario:** Es un rango de direcciones de memoria donde los programas comunes están restringidos para ejecutarse, con el fin de que un programa no pueda acceder a la memoria o a cualquier recurso de otro programa.

  En este modo la CPU solo puede acceder a la memoria etiquetada con derechos de acceso de espacio de usuario, solo mediante llamadas al sistema esta puede comunicarse con el espacio del kernel.

  Cuando un proceso realiza una llamada al sistema, se envía al kernel una interrupción software que activa el modo privilegiado, cuando este finaliza el proceso vuelve al espacio de usuario.

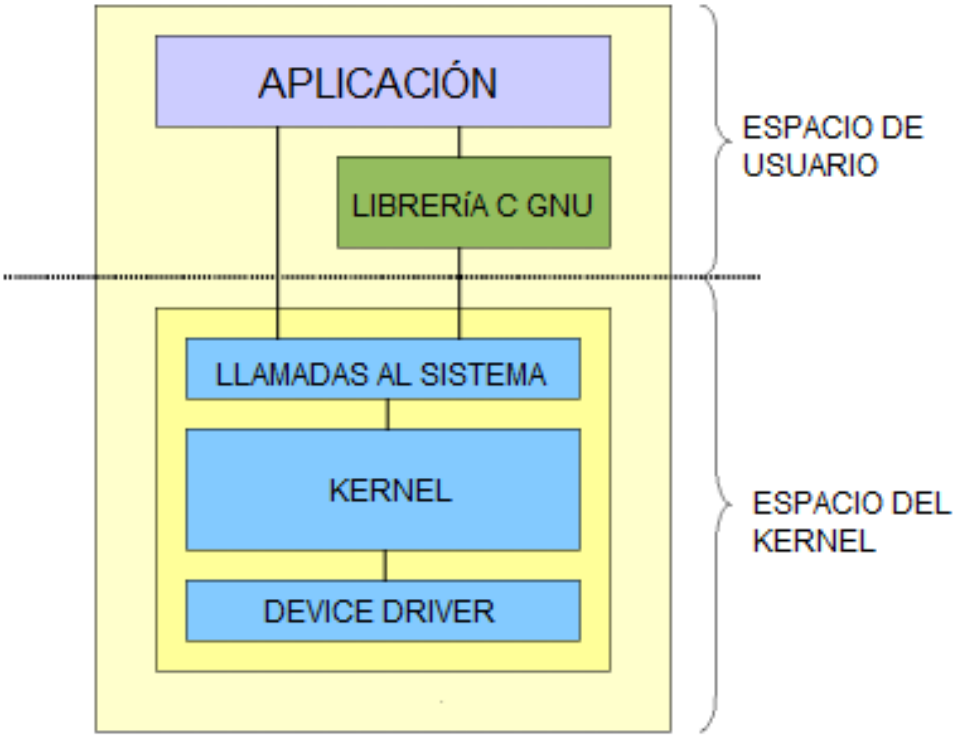



*Ilustración 3. Estructura Linux*

## 3.3.- Tipos de controladores dispositivos en Linux.

En Linux podemos clasificar los controladores de dispositivos en dos líneas, controladores de dispositivos de plataforma y controladores de dispositivos de caracteres, este último es el que se ha utilizado para este proyecto por lo que nos extenderemos más en él.

### 3.3.1.- Controlador de dispositivo de bloques.

Un dispositivo de bloque (Block device), es un dispositivo de almacenamiento de datos informáticos que admite la lectura y opcionalmente la escritura de datos en boques. Estos bloques generalmente son de 512 bytes o un múltiplo de éste.

Los drivers de dispositivos de bloques proporcionan un único procedimiento *request (),* que se usa tanto para leer como para escribir.

Existen procedimiento genéricos *block_read () y block_write (),* que llaman al procedimiento request () indicándole la acción que debe realizar.

Esta función request (), es de tipo void y no toma ningún argumento, sino que observa una cola de solicitudes de entrada/salida y las procesa una a una. Puede ser invocada por interrupciones o sin interrupciones:

- Si es invocada sin interrupciones, el procedimiento procesa solicitudes hasta que la cola se quede vacía.

- Si el dispositivo está controlado por interrupciones, el procedimiento request se programará como una interrupción y será el manejador de interrupciones el que indicará cuando debe procesar la solicitud.

### 3.3.2.- Controlador de dispositivos de caracteres.

El dispositivo de caracteres es el controlador más básico en la fuente del kernel de Linux, permiten transferir datos, por medio de caracteres, desde el espacio de usuario al espacio del kernel, en forma de flujo, al igual que un puerto serie.

Este expone las funcionalidades y propiedades de un dispositivo mediante un archivo en el directorio */dev* por el cual se pueden intercambiar datos el dispositivo y la aplicación.

Estos dispositivos se representan en el kernel como una instancia de *struct cdev,* que se encuentran definidas en *include/Linux/cdev.h.*

En otra estructura, llamada estructura de archivos, se definen los métodos por los cuales el kernel expone las capacidades del dispositivo al espacio de usuario, mediante las llamadas al sistema relacionadas con archivos (read, write, open, reléase, select, etc.).

Un dispositivo de caracteres es identificado por un número de 32 bits que se encuentra en el kernel, este número se divide en dos partes:

- **Mayor:** Contiene 12 bits, se utiliza para identifica al dispositivo.

- **Menor:** Contiene 20 bits, se suele utilizar como un índice para una matriz de una lista local de dispositivos, ya que un controlador puede manejar más de un dispositivo del mismo tipo.

Estos números pueden ser asignados de dos formas diferentes:

- **Estáticamente:** Asignar un número mayor que no esté siendo utilizado por otro dispositivo, para ello se utiliza la función *register_chrdev_region ().*

- **Dinámicamente:** El kernel es el que se encarga de asignar en este caso un número de dispositivo válido, la función utilizada para ello *sería alloc_chrev_region ().*

En los dos casos los métodos devuelven 0 si se asigna el número correctamente o un código de error negativo en caso de error.

Las operaciones de archivo más utilizadas para este tipo de dispositivos son las siguientes:

- **Open.** Permite abrir el dispositivo asignado recursos.
  Su prototipo es:
  *Int open (struct inode * i, struct file *f);*
  Descripción:
  - Llamada cuando el espacio de usuario abre el archivo del dispositivo.
  - Inode es una estructura que identifica de manera única un archivo en el sistema.
  - File es una estructura creada cada vez que se abre un archivo.

- El controlador utiliza este método para realizar cualquier inicialización, en preparación para operaciones posteriores.

- **Release.** Permite cerrar el dispositivo liberando recursos.
  Su prototipo es:
  *Int release (struct inode *i, struct file *f);*
  Descripción:
  - Llamada cuando se cierra el archivo desde el espacio de usuario.
  - Es la función contraria a open (), deshace todas las tareas realizadas por open ().

- **Read.** Permite leer los datos que provienen del dispositivo.
  Su prototipo es:
  *Ssize_read(struct file *file, __user char *buf, size_t size, loff_t *off);*
  Descripción:
  - Usada cuando el espacio de usuario quiere leer el dispositivo.
  - Escribe como máximo el tamaño el tamaño “size” en el buffer buff y actualiza la siguiente posición en el archivo off.
  - “file” es un puntero a la misma estructura que se le pasa a open ().
  - Devuelve el número de bytes leídos.

- **Write.** Permite escribir datos en el dispositivo.
  Su prototipo es:
  *Ssize_t write(struct file *file, __user char *buf, size_t size, loff_t *off);*

Descripción:

- Usada cuando el espacio de usuario quiere escribir en el dispositivo.
- Es la contraria a la función read ().
- Actualiza off y devuelve el número de bytes escritos.

- **ioctl.** Permite consultar las estadísticas del dispositivo y pasar los parámetros de configuración a este.
  **Su** prototipo es:
  *Static long ioctl(struct file *file, unsigned int cmd, unsigned long arg);*
  Descripción:
  - Asociada con la llamada al sistema ioctl.
  - Permite extender la capacidad de los drivers más allá de read/write.
  - Cmd es un número para identificar la operación.
  - Arg son argumentos opcionales, pueden ser números enteros, direcciones, etc.

# 4.- SYSTEMD

## 4.1.- Introducción.

Systemd es un sistema y administración de servicios, escrito por Lennart Poettering y publicado como software libre de código abierto, que se diseñó para el núcleo de Linux y programado exclusivamente para la API de éste, con el fin de sustituir el sistema de inicio (init) heredado de los sistemas operativos UNIX System V y Berkely Software Distribution para unificar los comportamientos de servicios y configuraciones básicas en todas las distribuciones.

Al ser el primer proceso en ejecutarse en el espacio de usuario en el arranque de Linux, es el proceso padre de todos los procesos hijos del espacio de usuario.

## 4.2.- Características.

Systemd ofrece las siguientes características:

- Utiliza socket para ofrecer una paralelización agresiva y poder acelerar el arranque iniciando más procesos en paralelo, para ello crea todos los sockets para todos los demonios en un solo paso en el sistema de inicio (init) y en un segundo paso ejecuta a la vez todos los demonios.

- Mantiene puntos de montaje y auto montaje.

- Ofrece inicio de demonios bajo demanda, realiza el seguimiento y rastreo de procesos utilizando cgroups, el cual es una característica de Linux que aísla, limita y representa los recursos usados de una colección de procesos.

- Soporta copia instantánea y restauración de volúmenes (snapshot) y la restauración de estado de sistemas.

- Implementa un elaborado servicio lógico de control transaccional basado en la dependencia.

## 4.3.- Tipos de unidades.

Systemd se basa en la noción de unidades compuestas de un nombre, tipo y coincidencia de un archivo de configuración, existen siete tipos de unidades:

1. **Service***:* Demonios que pueden ser iniciados, detenidos, reiniciados o recargados.

2. **Mount***:* Unidad que encapsula un punto de montaje en la jerarquía del sistema de archivos.

3. **Automount***:* Unidad que encapsula un punto de montaje automático en la jerarquía del sistema de archivos, cada unidad automount tiene una unidad mount correspondiente.

4. **Target***:* Se utiliza para para la agrupación lógica de unidades, no hace nada por sí misma, sino que hace referencia a otras unidades.

5. **Snapshot***:* Similar a las unidades target, su único propósito es hacer referencia a otras unidades.

6. **Device***:* Encapsula un dispositivo en el árbol de dispositivos de Linux. Si un dispositivo está marcado por las reglas de udev, se expondrá como una unidad device en Systemd.

7. **Socket***:* Encapsula un socket en el sistema de archivos o en internet, cada unidad de este tipo tiene una unidad de servicio correspondiente, que se inicia si la primera conexión entra en el socket.

# 5.- PANTALLA OLED CON CONTROLADOR SSD1306.

## 5.1.- Características.

La pantalla Oled de 0,96 pulgadas, una matriz de 128x64 pixeles monocromo, basada en el controlador ssd1306 puede funcionar con un rango de voltaje de 2.2 voltios a 5.5 voltios, el interfaz puede darse por i2c o Spi, en este proyecto se ha utilizado el interfaz i2c.

La dirección para comunicarse por i2c con la pantalla utilizada en este caso es 0x3C (Esto lo hay que verificarlo por el fabricante para cada pantalla).

## 5.2.- Organización de la memoria interna de su Frame Buffer.

El controlador de la pantalla internamente organiza en una memoria de 1KB los 128x64 píxeles que muestra por pantalla, esta memoria está organizada en 128 columnas de 8 filas de 8 bytes cada una.

Para transferir la información a la memoria de la pantalla se establecen dos punteros, uno índica la fila y el otro la columna donde se van a escribir los bytes. La información se envía de forma que los contadores se autoincrementen internamente por el controlador, para así, ir rellenando la memoria de la pantalla con los bytes enviados sucesivamente mediante el bus i2c.

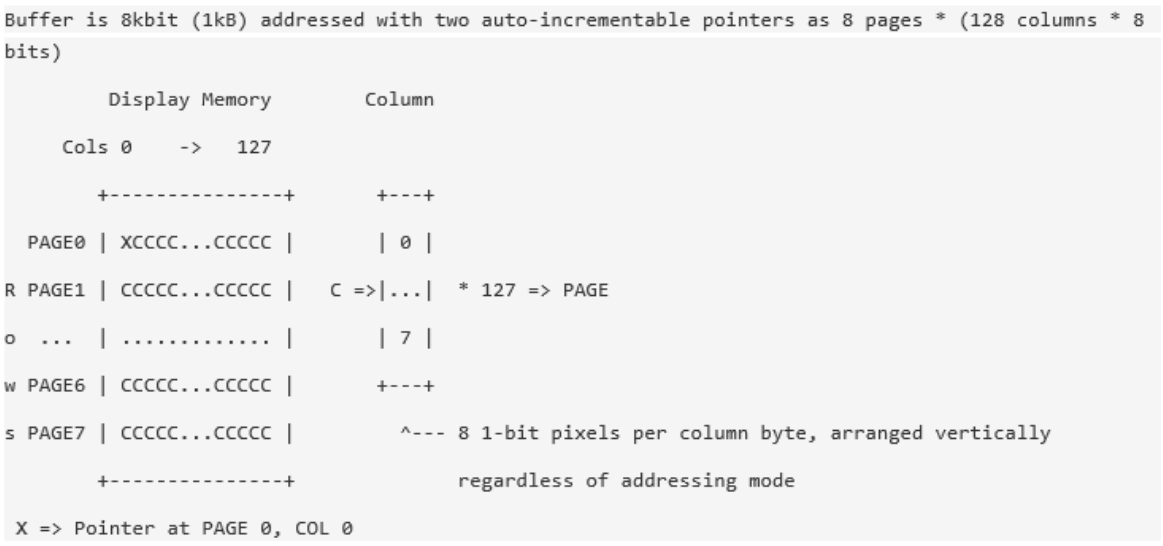



*Ilustración 4. Organización de la memoria de la pantalla*

## 5.3.- Fuente y memoria en driver.

Para pintar los caracteres en la pantalla se ha utilizado una fuente 8x8 con todos los caracteres imprimibles de ASCII, sacada de Commodore 64 8x8 *(http://kofler.dot.at/c64/*) y se ha volteado y reordenado con el programa que se encuentra en (*https://www.min.at/prinz/o/software/pixelfont/)* para que se pueda escribir directamente en el formato por columnas de la memoria de la pantalla.

Con esta fuente se puede usar la pantalla como una matriz de 16x8 caracteres de 8x8 ya que la misma fuente ya incorpora un pixel de separación entre caracteres.

Así ya podríamos escribir texto en el dispositivo que nuestro driver trasladaría a un buffer de texto de 16x8 que luego se traduce a un Frame buffer con un bitmap de 128x64 usando la matriz de fuentes y mandarlo a la pantalla a través de i2c.

# 6.- DEVICE TREE

## 6.1.- Introducción.

El Device Tree es una estructura de árbol con estilo de formato JSON donde se encuentra la descripción del hardware, cada dispositivo y sus propiedades es representado por un nodo, para que el núcleo del sistema operativo pueda administrar y usar esos componentes.

Éste se originó a partir de un estándar (Open Firmware), respaldado por compañías informáticas con el fin de definir interfaces para sistemas de firmware de computadora.

## 6.2.- Conceptos básicos sobre Device Tree.

- Los tipos de datos más comunes son:
    - Cadenas de texto, representada por comillas dobles. Se puede crear listas de cadenas, separadas por comas.
    - Enteros sin signo de 32 bits, delimitados por corchetes angulares “<>”.
    - Los datos Booleanos son una propiedad vacía, el valor depende de si la propiedad está o no.

- Cada nodo debe tener un nombre, el cual se indica de la siguiente forma, *<nombre> [@ <dirección>]*

  Donde:
    - <nombre>: Cadena que puede tener hasta 31 caracteres.
    - @<dirección>: Solo si el nodo representa a un dispositivo direccionable, en ese caso, debe ser la dirección utilizada para acceder al dispositivo.

- Los nodos pueden tener una etiqueta, que identifica a éste con un nombre único, ese nombre es transformado por el compilador en un número de 32 bits, estas se pueden usar para referirse a un nodo ya que son exclusivas de cada nodo.

- Un identificador de puntero (phandle) es un valor de 32 bits asociado con un nodo que se utiliza para identificar a este de manera única, de modo que se puede hacer referencia al nodo desde una propiedad en otro nodo, la sintaxis sería la siguiente:

  *<& label>*

  (Apunta al nodo cuya etiqueta es label).

- Para encontrar un nodo rápidamente, el kernel de Linux utiliza los alias, estos se encuentran en un nodo, que se puede ver como una tabla de búsqueda. Se puede utilizar la función *find_node_by_alias ()* para encontrar un nodo dado su alias

  .

## 6.3.- Compilación Device Tree.

El Device Tree tiene dos formas, la forma de texto conocida como DTS, es el fichero que creas con toda la información necesaria para el dispositivo con formato JSON y la forma binaria que representa al fichero ya compilado, este último tiene extensión DTB.

La herramienta usada para compilar un fichero de Device Tree se llama “Device Tree Compiler “(dtc), esta nos permite:

- Compilar un Device Tree específico.

- Compilar todos los Device Tree de una arquitectura específica.

- Dado un archivo compilado (dtb), realizar la operación inversa y extraer el archivo de origen (dts).

## 6.4.- Device Tree Overlay.

Cuando nos encontramos con un sistema como Raspberry Pi que admite accesorios completamente opcionales, cada configuración posible requiere un Device Tree para describirla, para ello se utiliza un Device Tree parcial.

Se construye un árbol completo tomando un Device Tree básico y se le añaden los elementos opcionales, estos elementos se llaman Overlay.

Este es el método que se ha utilizado para definir en el kernel el módulo cargable que se ha realizado, en el desarrollo del proyecto se verá con más detalle.

# 7.- DESARROLLO Y MANUAL DE PROGRAMACIÓN.

## 7.1.- Descripción General.

Como se introdujo al principio de esta memoria, el objetivo de este trabajo ha sido el estudio de la documentación del kernel de Linux relativa a la integración de drivers de dispositivos, sobre el bus de comunicación i2c, además del desarrollo completo del driver, la preparación de un método de compilación e instalación automatizado, el desarrollo de una pequeña librería C para el uso del dispositivo y el diseño de un servicio de Linux (Systemd) para mostrar información en el arranque del sistema.

Todo ello con el fin crear un manual de programación que sirva de guía para el desarrollo de drivers de dispositivos similares y poder utilizarlo en la asignatura del grado de ingeniería de Computadores, Diseño de sistemas operativos.

## 7.2.- Manual de programación

### 7.2.1.- Desarrollo del driver.

En este apartado se va a explicar paso a paso el desarrollo seguido para la programación del driver para una pequeña pantalla Oled conectada al bus i2c de una Raspberry Pi.

#### 7.2.1.1.- Organización.

El driver se divide en dos archivos, un archivo de cabecera con extensión .h en el cual se encuentran la estructura que representa a un dispositivo i2c que índica su nombre, el cliente i2c y el dispositivo; las cabeceras de las funciones que se implementan en el fichero .c y dos matrices, una que representa al buffer de la pantalla y otra con el tipo de fuente para poder pintar en la pantalla.

El fichero con extensión .c es el que contiene la implementación completa del driver, desde las funciones propias necesarias para la instalación, desinstalación y registro del driver hasta las funciones de configuración y uso para la pantalla; la estructura que define al driver i2c, con sus funciones

principales, probe encargada de inicializar el driver y remove encargada de la desinstalación del driver; y otra estructura con las funciones para comunicarse con el dispositivo una vez inicializado, en este caso serían ioctl y write.

Cuando el sistema detecta que un dispositivo conectado tiene el mismo id que un módulo cargable, salta la función probe que se encarga de registrar el driver i2c y de mandar al dispositivo la secuencia de bytes necesarios para su inicialización.

Una vez registrado e inicializado, tenemos el dispositivo como un fichero montado en /dev, cada vez que se escribe en ese fichero se llama a la función write, que es la encargada de transformar los caracteres a píxeles (bytes) utilizando para ello la matriz de la fuente del fichero de cabecera y escribirlos en el buffer que representa a la pantalla, para ser mandado este mediante las funciones del kernel de i2c y el cliente i2c, al dispositivo para ser pintados (*como se explica en la página 18* ). Sí se le envía al dispositivo el código correspondiente al ioctl, este se encarga de borrar el buffer por completo y mandarlo a la pantalla, por lo que esta quedaría borrada.

Cuando se desinstala el driver, el sistema hace que salte la función remove del driver, encargada de eliminar el driver del sistema y desmontar el fichero que corresponde al dispositivo.

#### 7.2.1.2.- Archivo de cabecera ssd1306_oled.h

En este apartado se van a explicar una a una todas las definiciones que se encuentran en él, que se han utilizado para la programación del driver.

- Buffer de memoria que representa a la pantalla oled, utilizado para poder mostrar caracteres por la pantalla. (*Página 18*)
- Zona de memoria que contiene la fuente, (mayúsculas, minúsculas, números, etc.) (*Página 18*)
- Definición de la estructura para un dispositivo i2c, la cual contiene:
    - Un puntero hacía el dispositivo.
    -

- Una cadena de caracteres para indicar el nombre del dispositivo.

- La estructura de cliente i2c

- La estructura miscdevice que es la utilizada para asignar el número menor al dispositivo, como se hace referencia en el apartado *(página 11*).

```
struct ssd1306_oled {
    struct device *dev;
    char devname[11];
    struct i2c_client *my_client;
    struct miscdevice miscdev;
};
```

- Estructura que asigna un id al dispositivo.

```
static const struct i2c_device_id ssd1306_i2c_id[] = {
    {"ssd1306_oled", 0},
    { }
};
```

- Por último, se encuentran las cabeceras de todas las funciones que implementa el driver.

#### 7.2.1.3.- Archivo ssd1306_oled.c

Este archivo contiene toda la implementación del driver, al igual que con la cabecera, se va a explicar cada una de las funciones que están implementadas.

- En primer lugar, se encuentran incluidas todas las cabeceras necesarias para la implementación de las funciones.

- Se declaran todas las macros que se van a necesitar, para la inicialización de la pantalla, ioctl, scrolll, …

- ***static void clear_buffer ()***. Función que se encarga de borrar la pantalla, para ello asigna un cero a todas las posiciones de memoria del buffer.

- ***static void ssd1306_command ().*** Función que se encarga de ejecutar el protocolo para escribir el byte que se le pasa, para ello, se utiliza la función:

```
i2c_smbus_write_byte_data(my_oled->my_client, command, value);
```

  A la que se le pasa:
  - Un cliente i2c, en este caso el del dispositivo registrado.
  - Un comando, para indicar al esclavo como debe interpretar los datos, para ello se le envía el byte de control, que consiste principalmente en el byte *Co* y *D/C,* seguidos de seis “0”.
    - Se le asigna a *Co* un “0”, para indicar que la siguiente transmisión de información contendrá solo bytes de datos.
    - Se le asigna a *D/C* un 0, para definir el siguiente byte de datos como un comando.
  - El byte que se va a escribir.

- ***static void ssd1306_writedatablock ().*** Función que recibe un puntero a un array de bytes y su tamaño y se encarga de ejecutar el protocolo para escribir un bloque de datos, utilizado la función:

```
i2c_smbus_write_i2c_block_data(my_oled->my_client, command, length, values);
```

  A la que hay que pasarle:
  - Un cliente i2c, en este caso el del dispositivo registrado.

  - Un comando, para indicar al esclavo como debe interpretar los datos, para ello se le envía el byte de control, que consiste principalmente en el byte *Co* y *D/C,* seguidos de seis “0”.
    - Se le asigna a *Co* un “0”, para indicar que la siguiente transmisión contendrá solo bytes de datos.
    - Se le asigna a *D/C* un 1, para definir el siguiente byte de datos como un dato que se almacenará en GDDRAM.
  - La longitud del bloque que se va a transmitir.
  - El array de bytes que se van a transmitir.

- ***static void display ().*** Función que realiza el protocolo para escribir en pantalla, indicándole desde hasta donde se va a escribir, utilizando la función *ssd1306_command (),* y escribiendo lo que haya en el buffer que representa a la pantalla en bloques de 16 utilizando la función *ssd1306_writedatablock ().*

- ***static void init_sequence ().*** Función que realiza el protocolo para la inicialización de la pantalla para ello utiliza las macros declaradas arriba y la función *ssd1306_command (),* para enviar el byte necesario a la pantalla en cada caso.

- ***void printchar ().*** Función que escribe el carácter que recibe en la posición en la que nos encontramos del buffer.

- ***void printoled ().*** Función que recibe una cadena de caracteres y su tamaño y se encarga de pasar esos caracteres a la función *printchar ()* y poner la posición del cursor en el sitio correcto, para imprimir la cadena en la pantalla oled.

- ***static ssize_t ssd1306_write ().*** Como se indica en la descripción de los drivers de dispositivos de caracteres (*página 13*), está función es una función propia de estos, y es la encargada de leer los datos desde

el espacio de usuario y pasarlos al espacio del kernel para ser procesados, en este driver, esta función se encarga de leer

las cadenas de caracteres que se quieren imprimir en la pantalla oled y pasarla a la función ***printoled (),*** para que así sea.
El prototipo de esta función es el siguiente:

```
static ssize_t ssd1306_write(struct file *file, const char __user *buf,
                             size_t count, loff_t *ppos){
```

Donde:

- *file → puntero al fichero que representa al dispositivo.
- *buf → puntero al buffer que contiene los datos en el espacio de usuario.
- count → tamaño del buffer del espacio de usuario.
- *ppos → puntero que índica la posición donde comienzan los datos.

Para copiar los datos del espacio de usuario al espacio del kernel se utiliza la siguiente función:

```
if (copy_from_user( &cadena, buf, count )) {
    return -EFAULT;
}
```

- ***static long ssd1306_ioctl ().*** Función específica de este tipo de dispositivos, ya descrita en (*página 14*), que nos permite un control sobre el hardware, en este caso el borrado de la pantalla.
  Su prototipo es:

```
static long ssd1306_ioctl(struct file *file, unsigned int cmd, unsigned long args){
```

Donde:

- *file → puntero al fichero que representa al dispositivo.
- cmd → número que hace referencia a la función que se desea realizar sobre el hardware, ya que está función se suele implementar mediante un *switch.*

- ***struct ssd1306_oled ssd1306_i2c_register_hardware ().*** Función a la que se le pasa como parámetros un puntero a un dispositivo y un puntero a un cliente i2c, y se encarga de reservar memoria para la estructura, inicializar la estructura con todos los parámetros necesarios para registrar el dispositivo, llamar a la función que se describe más abajo para añadir el dispositivo *int ssd1306_i2c_add_device ()* y devolver la estructura.

- ***int ssd1306_probe ().*** Esta función es la primera a la que se llama cuando se instala el driver en el sistema, recibe un puntero a una estructura de un cliente i2c y un puntero al id del dispositivo, se encarga de llamar a la función para registrar el dispositivo, inicializar lo necesario en este caso para utilizar la pantalla, como la inicialización de la pantalla y situar el cursor en la posición (0,0), para empezar a escribir desde ahí.

- ***int ssd1306_i2c_add_device ().*** Función a la que se llama cuando se quiere desinstalar el driver, se le pasa un puntero a una estructura previamente inicializada desde la función *struct ssd1306_oled ssd1306_i2c_register_hardware (),* y esta se encarga de registrarlo, con la

  siguiente función del kernel de Linux *misc_register (),* si esta función devuelve 0 el dispositivo se ha registrado correctamente, si devuelve otro valor se manda un mensaje de error porque no se puede registrar el dispositivo.

- ***int ssd1306_remove ().*** Función que recibe un puntero a una estructura de un dispositivo i2c, en este caso a la pantalla oled, borra la pantalla, comprueba que dicho dispositivo esta registrado, si es así, lo elimina

- utilizando la función del kernel *misc_deregister (),* y libera la memoria reservada para dicha estructura.
- Se define la estructura del driver i2c, en la cual se indica el nombre del driver, cual es la función probe y remove, y el id del dispositivo i2c.

```
MODULE_DEVICE_TABLE(i2c, ssd1306_i2c_id);

static struct i2c_driver ssd1306_i2c_driver  = {
    .driver = {
        .owner = THIS_MODULE,
        .name  = "ssd1306_oled",
    },

    .probe    = ssd1306_probe,
    .remove   = ssd1306_remove,
    .id_table = ssd1306_i2c_id,

};
```

- ***static int __init ssd1306_i2c_init ().*** Esta función se encarga de detectar cuando se quiere instalar un driver de este dispositivo e invoca a la función *probe* para que esta se encargue.

```
static int __init  ssd1306_i2c_init(void){

    return i2c_add_driver(&ssd1306_i2c_driver);

}
```

- ***static void __exit ssd1306_i2c_cleanup ().*** Esta función se encarga de detectar cuando se quiere desinstalar un driver de este dispositivo e invoca a la función *remove* para que esta se encargue.

```
static void __exit ssd1306_i2c_cleanup(void){

    i2c_del_driver(&ssd1306_i2c_driver);

}
```

### 7.2.2.- Definir módulo en Device Tree Overlay.

7.2.2.1.- Archivo Device Tree ssd1306-i2c.dts.

Para registrar el dispositivo en el Device Tree, se ha creado el fichero llamado ssd1306-i2c.dts el cual tiene la información necesaria para registrar un dispositivo i2c.

```
/dts-v1/;
/plugin/;

/{
    compatible = "brcm,bcm2709";

    fragment@0 {
        target = <&i2c1>;
        __overlay__ {
            #address-cells = <1>;
            #size-cells = <0>;
            status = "okay";

            oled: oled@3c {
                    compatible = "ssd1306_oled";
                    reg = <0x3c>;
                    status = "okay";
            };
        };
    };

};
```

Donde:

- Se le indica que el target se encuentra en i2c1.

- Se le indica que la dirección del dispositivo es la dirección *0x3c,* esto se puede ver conectando la pantalla Oled y ejecutando en terminal "i2cdetect -y 1".
- Se le indica que es compatible con ssd1306_oled, para que coincida con el nombre de nuestro driver.

#### 7.2.2.2.- Compilación Device Tree.

Para compilar el Device Tree utilizamos la siguiente línea en la terminal de comandos dentro de donde se encuentre el archivo *.dts.*

```
dtc -O dtb -o ssd1306-i2c.dtbo -b 0-@ ssd1306-i2c.dts
```

La salida nos da un archivo con la extensión *.dtbo,* que es el archivo .dts compilado y a parte se indica que es un dispositivo para el Device Tree Overlay

#### 7.2.2.3.-Introducir el fichero .dtbo en Device Tree Overlay.

- Copiamos el archivo ssd1306.dtbo en el directorio *"/boot/overlay"*
- El archivo config.txt, que se encuentra en el directorio "*/boot/config.txt"*, lo modificamos introduciendo una nueva línea "*dtoverlay=ssd1306-i2c"* y descomentando la línea *"dtparam=i2c_arm=on".*

### 7.2.3.- Compilación driver.

Para la compilación se ha realizado un Makefile para generar a partir de los archivos del driver, los archivos necesarios para poder cargarlos en el sistema.

```
MODULE=ssd1306_oled

KERNEL=`uname -r`
KERNEL_SRC=/lib/modules/${KERNEL}/build

obj-m += ${MODULE}.o

compile:
    @echo Usando kernel ${KERNEL}
    make -C ${KERNEL_SRC} M=${CURDIR} modules
```

## 7.2.4.- Instalación/desinstalación automatizada del driver.

### 6.2.4.1.- Desde Makefile.

Se ejecuta en la terminal *"make install"* o *"make unnistall"* cuando se desee, ya que han sido implementados.

```
install:
    sudo insmod ${MODULE}.ko
    dmesg | tail
    #echo ssd1306_oled 0x3c | sudo tee /sys/class/i2c-adapter/i2c-1/new_device
    sudo chgrp i2c /dev/ssd1306_oled
    sudo chmod go+rw /dev/ssd1306_oled


uninstall:
    sudo rmmod ${MODULE}
    dmesg | tail
```

La línea comentada en la parte del install se necesitaría si todavía no se ha añadido el dispositivo en el Device Tree, está lo registra en esa sesión para poder instalar el driver, como se supone que ya se ha añadido, se encuentra comentada.

### 7.2.4.2.- Con un servicio de Systemd.

Esta opción sería la más correcta, ya que se instala el driver en el arranque del sistema, para ello se ha realizado un script de Systemd

.

```
[Unit]
Description=Carga modulo para pantalla oled, crea /dev/ssd1306_oled

[Service]
Type=oneshot
ExecStart=/sbin/insmod /home/pi/pantalla_oled_ssd1306/driver/ssd1306_oled.ko
RemainAfterExit=true
ExecStop=/sbin/rmmod ssd1306_oled

[Install]
WantedBy=multi-user.target
```

Este script se debe cargar en el arranque del sistema, para ello ver las indicaciones que se dan en el siguiente apartado (**Cargar servicios Systemd al arranque del sistema**).

### 7.2.5.- Desarrollo de librería C para el uso de la pantalla.

Para poder usar la pantalla desde lenguajes de alto nivel, se ha creado una librería en C, que permite escribir y borrar la pantalla sin tener que saber cuál es el fichero que representa a dicho dispositivo, ni tener que saber cómo funciona este.

- Para escribir en pantalla, se ha creado una función llamada *oled_printf* basada en el funcionamiento de la función *printf* de C.
  Ejemplo para escribir por pantalla:

  ```
  oled_printf("Numero: %d\n", 12);
  ```

- Para borrar la pantalla se ha creado la función *oled_clean(),* que se basa en pasarle el código correspondiente mediante la función *ioctl* para el borrado de esta.

### 7.2.6.- Dirección IP de Raspberry por pantalla Oled.

Se ha realizado un script para imprimir por la pantalla Oled la dirección IP de la Raspberry pi.

```
#!/bin/bash

interfaz=$(ifconfig |grep eth0 |awk '{print $1}'|head -$i |tail -1)
ip=$(ifconfig |grep -A 1 eth0 |grep -v "eth0"| awk '{print $2}'|tr -d addr:|head -$head |tail -1)
echo            > /dev/ssd1306_oled
echo $interfaz  > /dev/ssd1306_oled
echo $ip        > /dev/ssd1306_oled
```

y un servicio de systemd para ejecutarlo en el arranque del sistema.

```
[Unit]
Description=Muestra direccion IP por pantalla Oled
After=ssd1306_oled.service network.target

[Service]
Type=oneshot
ExecStart=/bin/bash /home/pi/pantalla_oled_ssd1306/script_ip/script_ip_oled.sh
RemainAfterExit=true

[Install]
WantedBy=multi-user.target
```

En este servicio se ha añadido a *Unit* a parte de la descripción, After que indica los servicios de los que depende, para evitar que se inicie este servicio antes de que el driver este instalado y no pueda escribir en él, o antes de que la interfaz de red este activa.

Para ver como añadirlo al arranque ver siguiente apartado (**Cargar servicios Systemd al arranque del sistema)**.

### 7.2.7.- Cargar servicios de Systemd al arranque del sistema.

Para cargar los scripts de Systemd en el arranque del sistema, hay que seguir los siguientes pasos, para ellos suponemos que el archivo que queremos cargar es el siguiente *ssd1306_oled.service*.

- Copiar *ssd1306_oled.service* en el directorio:

  *"/lib/systemd/system"*

- Activar el servicio con el siguiente comando:

  *"sudo systemctl enable ssd1306_oled.service"*

- Al reiniciar el sistema, el servicio se ejecutará en el arranque.

# 8.- MANUAL DE INSTALACIÓN

Aunque durante el desarrollo de la memoria se han ido explicando cómo realizar algunos pasos, aquí se va a describir paso por paso como instalar todo lo necesario para que funcione el proyecto realizado.

Pasos para Descargar, compilar e instalar driver en el arranque del sistema.

- **Descargar proyecto**

  *> "git clone https://JoseCarlosNavarroGonzalez@bitbucket.org/JoseCarlosNavarroGonzalez/proyecto_pantalla_oled_ssd1306.git"*

- **Descargar headers**

  *> "sudo apt-get install-headers"*

- **Acceder a la carpeta *<u>/driver</u>* y compilar el módulo.**

  *> "make"*

- **Acceder a la carpeta /device_tree y compilar el archivo.**

  *> "dtc -O dtb -o ssd1306-i2c.dtbo -b 0-@ ssd1306-i2c.dts"*

- **Copiar el archivo que se ha creado, *ssd1306-i2c.dtbo,* en el directorio */boot/overlays.***

  *> "sudo cp ssd1306-i2c.dtbo /boot/overlays"*

- **Modificar archivo */boot/config.txt.***

  *Descomentamos → dtparam=i2c_arm=on*

  *Añadimos → dtoverlay=ssd1306-i2c*

- **Acceder a la carpeta */ssd1306_service_systemd_driver,* modificar la linea:**

  "*ExecStart=/sbin/insmod /home/pi/pantalla_oled_ssd1306/driver/ssd1306_oled.ko."*

  **Por el directorio dónde se encuentre el driver que se ha compilado anteriormente quedando al final:**

  *"ExecStart=/sbin/insmod "/*MI_DIRECTORIO*"/ssd1306_oled.ko."*

- **Copiar *ssd1306_oled.service* en el directorio */lib/systemd/system*.**

  *>" sudo cp ssd1306_oled.service /lib/systemd/system"*

- **Habilitar el servicio**.

  *>" sudo systemctl enable ssd1306_oled.service"*

- **Conectar la pantalla en los pines correspondientes.**
  - GND -> gnd.
  - VDD -> 3.3v.
  - SDK -> gpio03.
  - SDA -> gpio02.
- **Reiniciar Raspberry pi y tras el arranque se debería ver en la pantalla Oled:**

  "-PANTALLA OLED-"

# 9.- CONCLUSIÓN Y LÍNEAS FUTURAS.

## 9.1.- Objetivos cumplidos.

Tras una evaluación general del proyecto se puede afirmar que los objetivos del proyecto, planteados inicialmente, han sido cubiertos.

Se ha conseguido realizar el driver para una pantalla oled SSD1306, añadirlo al Device Tree Overlay y que este se instale en el arranque del sistema mediante un servicio de systemd, así como la realización de una librería en C, para el uso de la pantalla desde un lenguaje de más alto nivel y el desarrollo de otro servicio que nos muestra la ip de la Raspberry Pi sobre la pantalla.

## 9.2.- Líneas futuras.

Como líneas futuras para este trabajo, podrían ser ejercicios para la asignatura de diseño de sistemas operativos con los que se podría ampliar este proyecto.

Algunos de ellos podrían ser:

- Añadir un cursor parpadeante para indicar la posición de escritura en el display. Dentro de un timer o tasklet no se puede hacer uso de las funciones de comunicación i2c porque estás pueden bloquear y el kernel no lo permitirá, por lo que habría que usar desde la función del timer un work queue para ejecutar desde allí la comunicación i2c para hacer parpadear al cursor.

- Añadir más funciones ioctls:
    - Para mover el cursor (arriba, abajo, o a una posición dada).
    - Invertir el video y que pinte negro sobre blanco.
    - Cambiar el brillo/contraste de la pantalla.
    - Apagar y encender la pantalla.
    - Cambiar el tipo de fuente.

- Añadir un buffer de texto de más líneas que las que tiene la pantalla para hacer scrolll y poder recuperar el texto antiguo que desapareció por arriba.

- Añadir algún botón a los gpios, controlados por interrupciones para poder mover el cursor sobre la pantalla.

# 10.- REFERENCIAS

**https://es.wikipedia.org/wiki/I%C2%B2C**

**https://www.diarioelectronicohoy.com/blog/introduccion-al-i2c-bus**

**http://i2c.info/i2c-bus-specification**

**https://es.wikipedia.org/wiki/Systemd**

**https://www.freedesktop.org/wiki/Software/systemd/**

**Libro: Linux Device Drivers Development**

**datasheet: https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf**

**Commodore: http://kofler.dot.at/c64/font_01.html**

**https://elixir.bootlin.com/linux/v4.7/ident/i2c_smbus_write_i2c_block_data**

**REPOSITORIO TRABAJO:**

*"https://JoseCarlosNavarroGonzalez@bitbucket.org/JoseCarlosNavarroGonzalez/proyecto_pantalla_oled_ssd1306.git"*