<hay secciones a mitad que todavía no están procesadas y pulidas>
<la parte de ejercicios es muy interesante, está al final, no es necesario cubrir toda la teoría antes de proceder con los ejercicios>
<algunos diagramas están hechos en inglés pensando en una futura traducción al idioma>

< falta añadir actividades que alivien el "tocho" que supone este tema. Las que hay son interesantes pero hacen falta más>

Hardware en Linux y udev

En este documento se describen diferentes conceptos que giran alrededor de los siguientes puntos

- Organización y detección de dispositivos a nivel de hardware. Funciones de los buses PCI y USB
- Interacción entre el sistema operativo y el hardware de cara a la detección, identificación de hardware y posterior carga de drivers
- Fases y etapas del arranque y del reconocimiento del hardware.
- Recursos ofrecidos por el sistema operativo al administrador y usuarios para informar sobre las características del hardware
- Configuración de las respuestas del sistema a cambios en el hardware (UDEV)

Dado que este texto se destina a ser usado en ciclos de informática de Formación Profesional, se complementará con información histórica y soluciones ya anticuadas que ayuden a los alumnos a entender problemáticas ya superadas.

No se recomienda como texto introductorio a conceptos como sistema operativo, ejecución de procesos, elementos de hardware de un ordenador, sistemas de ficheros, etc. Más bien, el alumno debe tener claros esos conceptos para progresar en este documento adecuadamente. Durante todo el texto es conveniente ir verificando y observando con un computador aquello que es descrito aquí.

Este texto es ideal para el módulo de Fundamentos de Hardware de primero de ASIR, pero igualmente tiene gran parte de coincidencias con conceptos de los módulos de Implantación de Sistemas Operativos y Administración de Sistemas Operativos de ASIR, Montaje y Mantenimiento de primero de ciclo Medio, Sistemas informáticos en primero de DAW/DAM.

Documentación de referencia: enlace1 Enlace2

Índice:

GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE SECRETARÍA DE ESTADO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y UNIVERSIDADES
Unión Europea Fondo Social Europeo El FSE invierte en tu futuro

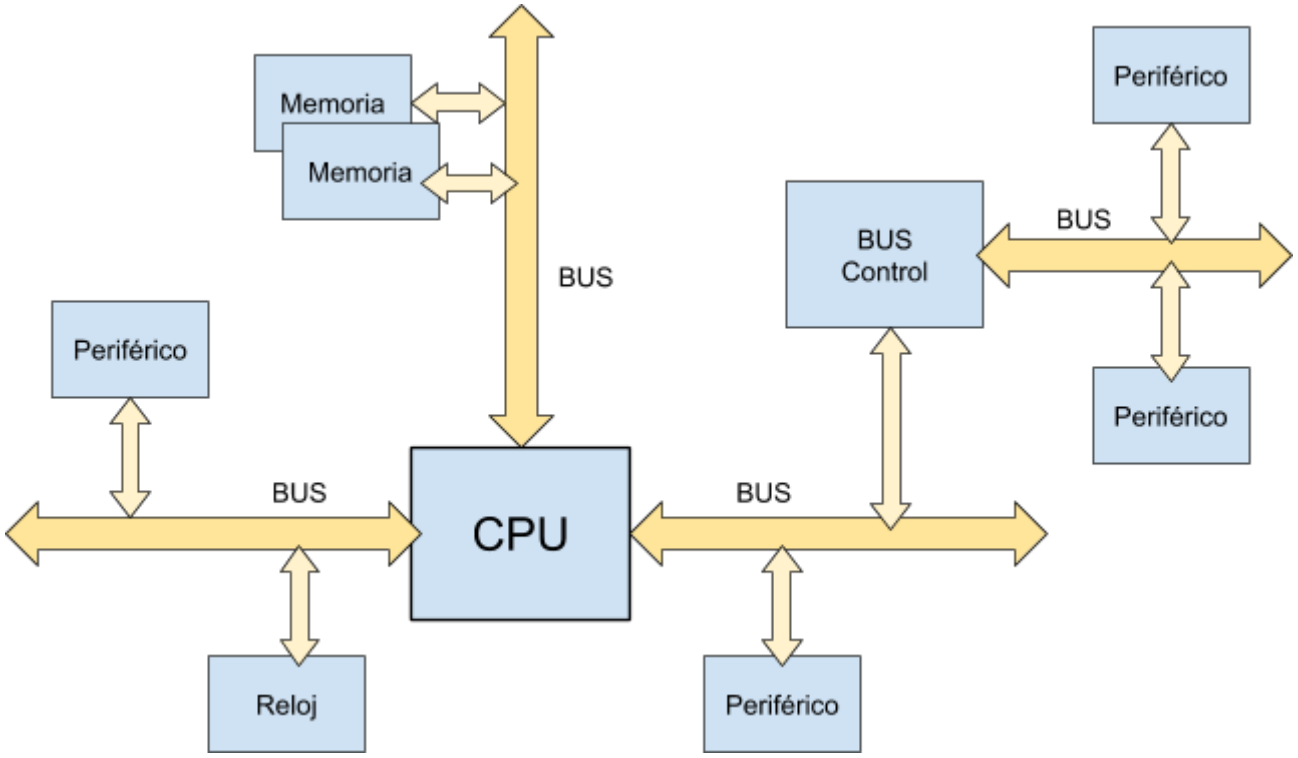
Índice:

GOBIERNO DE ESPAÑA MINISTERIO DE EDUCACIÓN, CULTURA Y DEPORTE SECRETARÍA DE ESTADO DE EDUCACIÓN, FORMACIÓN PROFESIONAL Y UNIVERSIDADES
Unión Europea Fondo Social Europeo El FSE invierte en tu futuro

Organización física del hardware

Aunque pensamos en el hardware periférico del ordenador como en el conjunto de tarjetas de red, gráfica, ratón, disco duro, teclado, etc. la realidad es que **el principio de todo el sistema de organización del Hardware está en los buses**.

Los buses son las líneas que interconectan distintas partes del computador.



http://www.makelinux.net/ldd3/chp-14-sect-4

Algunos buses son transparentes al sistema operativo; es decir, ni el sistema operativo, ni drivers, ni programas pueden percibir esos buses (por ejemplo el que une la CPU y Memoria). Con estos buses transparentes no se necesitan drivers ni nada similar. Pero otros buses son gestionados por software; es decir: algunos buses tienen su propio driver y son tan complejos como cualquier otro dispositivo. Y **el sistema operativo ve a esos buses como un dispositivo más**. Todo lo que entre y salga por el bus (datos, comandos, diagnósticos, modos, etc) es gestionado por el driver del bus. En este tema, los buses de este tipo que más trataremos serán el PCI (bus interno al ordenador) y el bus USB.

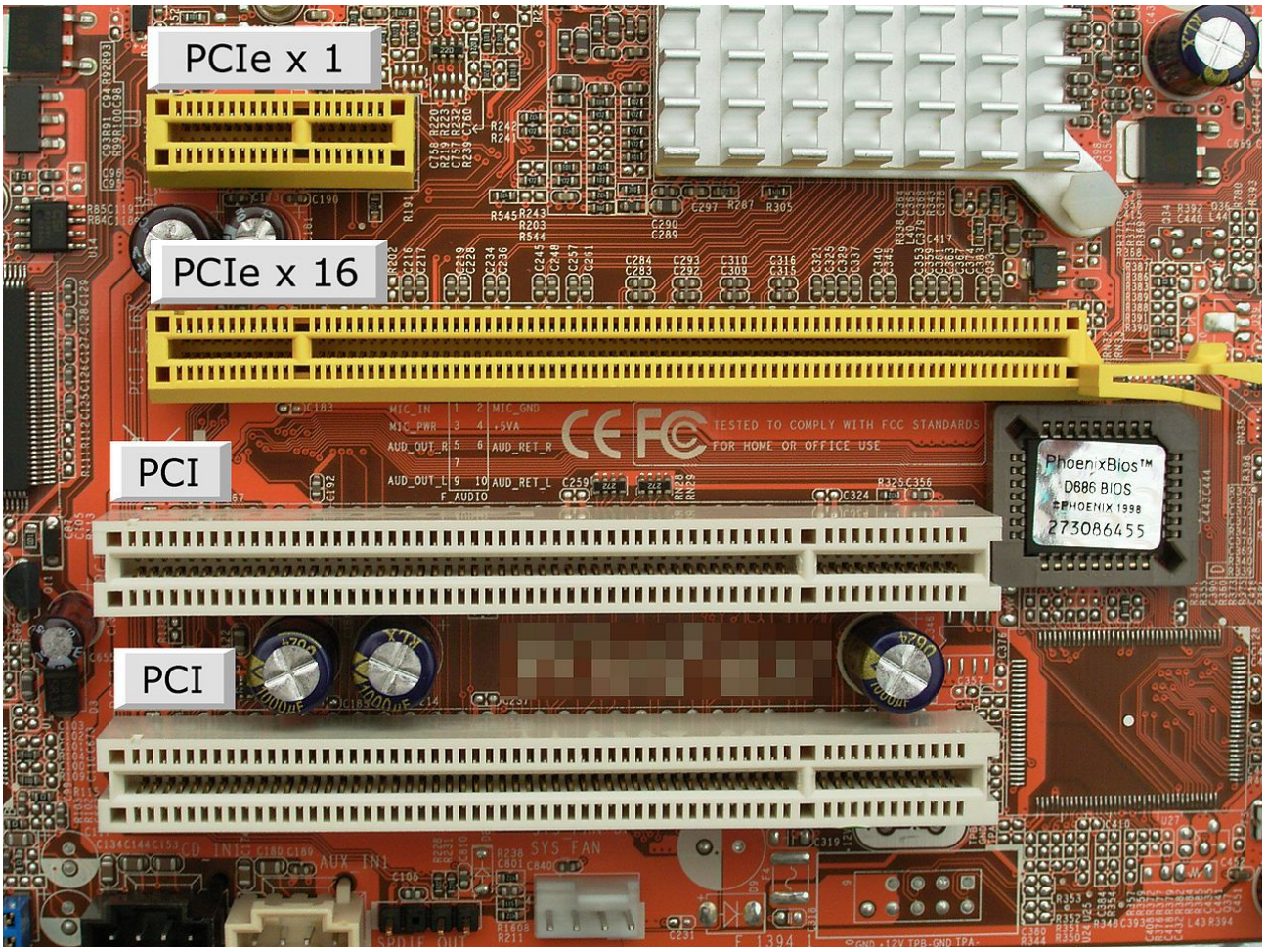
Más allá de los buses, El sistema operativo percibe y gestiona los periféricos como elementos conectados a algún bus. De hecho la mayoría de periféricos se identifica primitivamente por su conexión: bús y número de conexión dentro del bús. Por ejemplo, una tarjeta gráfica podría estar identificada como "`Bus PCI 0, conexión 7`". El tipo de dispositivo real (impresora, tarjeta de red, memoria usb, etc.) no es tan importante para el sistema operativo en un primer momento.

El bus pci :

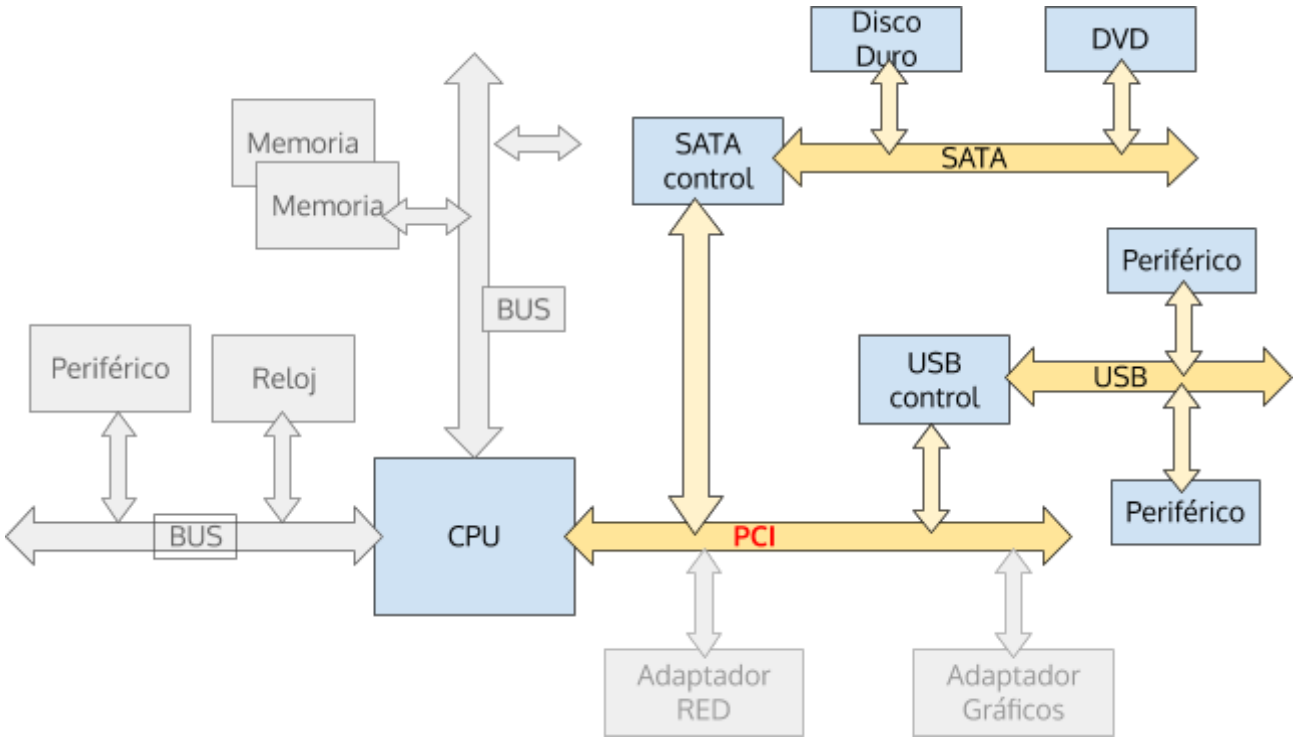
El bus pci es el más importante de un computador "PC" hoy en día. Es un bus que interconecta los dispositivos principales y básicos del computador (excepto la memoria y pocos más). Algunos de estos dispositivos son periféricos de entrada salida y otros dispositivos, buses, como por ejemplo el bus usb o sata.

El bus PCI es "interno" en el sentido en que los dispositivos están en la placa base o

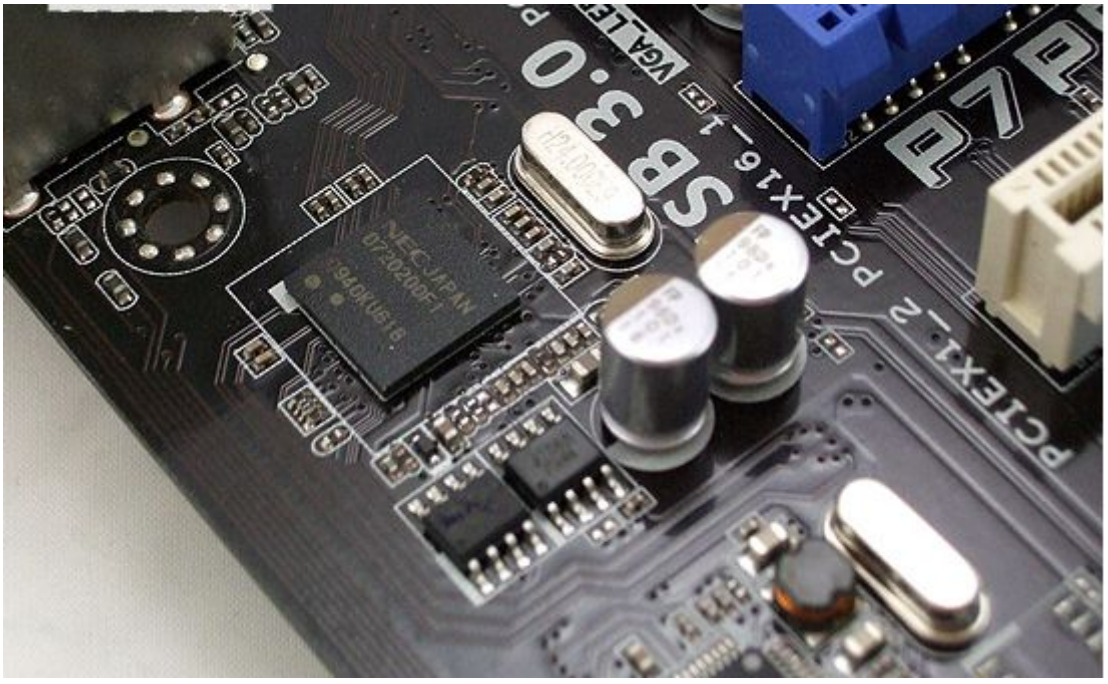
conectados directamente a zócalos pci. No es un bus para ampliar el computador con elementos externos a la "caja". Un placa base tiene slots para añadir hardware conectado a dicho bus como se ve en la siguiente imagen



El bus conecta además, circuitos (chips) y dispositivos existentes en la placa base (no sólo los que hay en los zócalos como los de la imagen anterior). Estos chips pueden a su vez controlar buses externos. Por ejemplo, el bus USB está subordinado a un dispositivo conectado al bus PCI ("el usb cuelga del pci"). Lo mismo ocurre con el bus SATA que conecta discos duros, DVD, etc.



Muy pocos dispositivos no están conectados al bus pci, y son los más antiguos. El resto está conectado de alguna u otra forma al bus pci. Por ejemplo, el controlador usb está empotrado en otro chip o aparece soldado a la placa base generalmente pero conectado al bus pci.

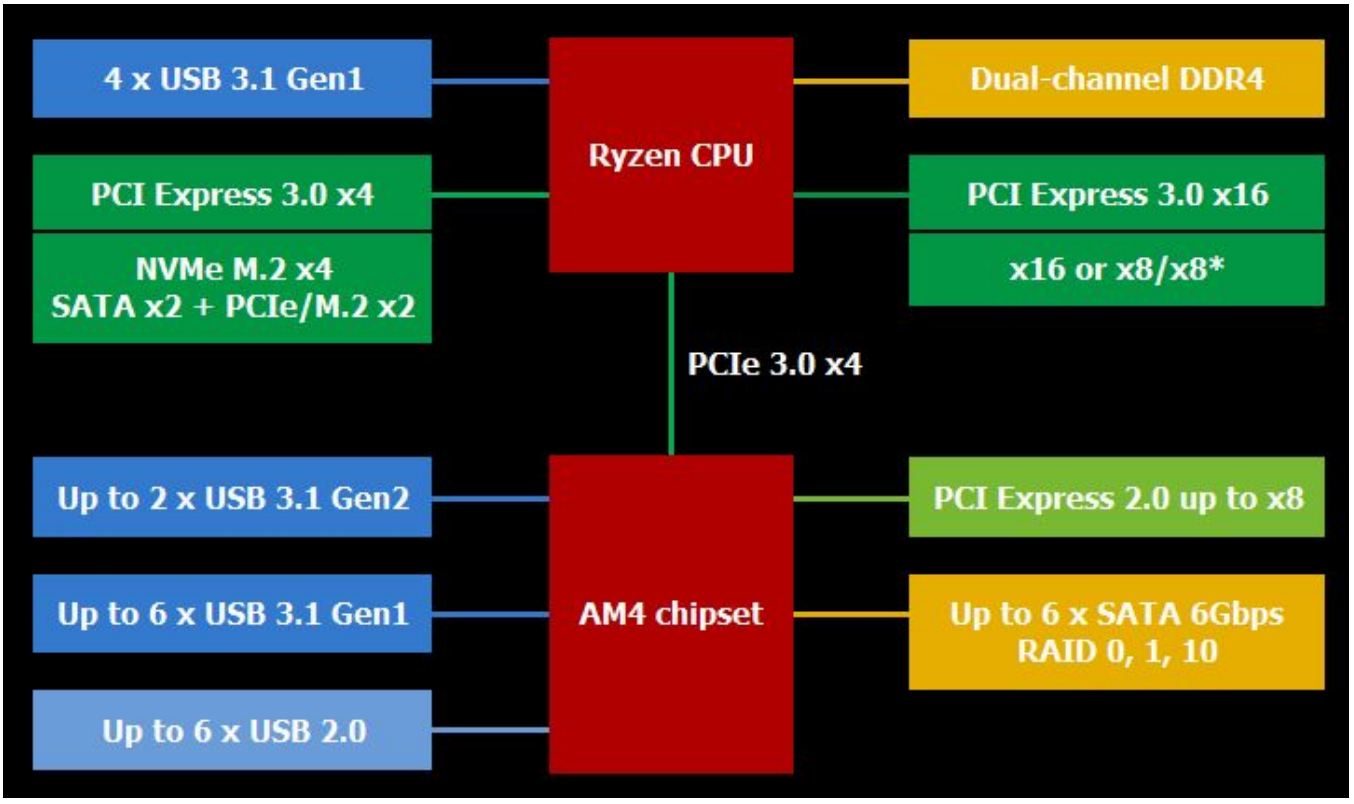


Controlador USB 3 en una placa base conectado a una línea pci en la propia placa

Sin embargo, el bus pci es poco conocido popularmente (a diferencia del USB) porque es interno y pese a que el usuario puede añadir hardware a los zócalos pci raramente lo hace.

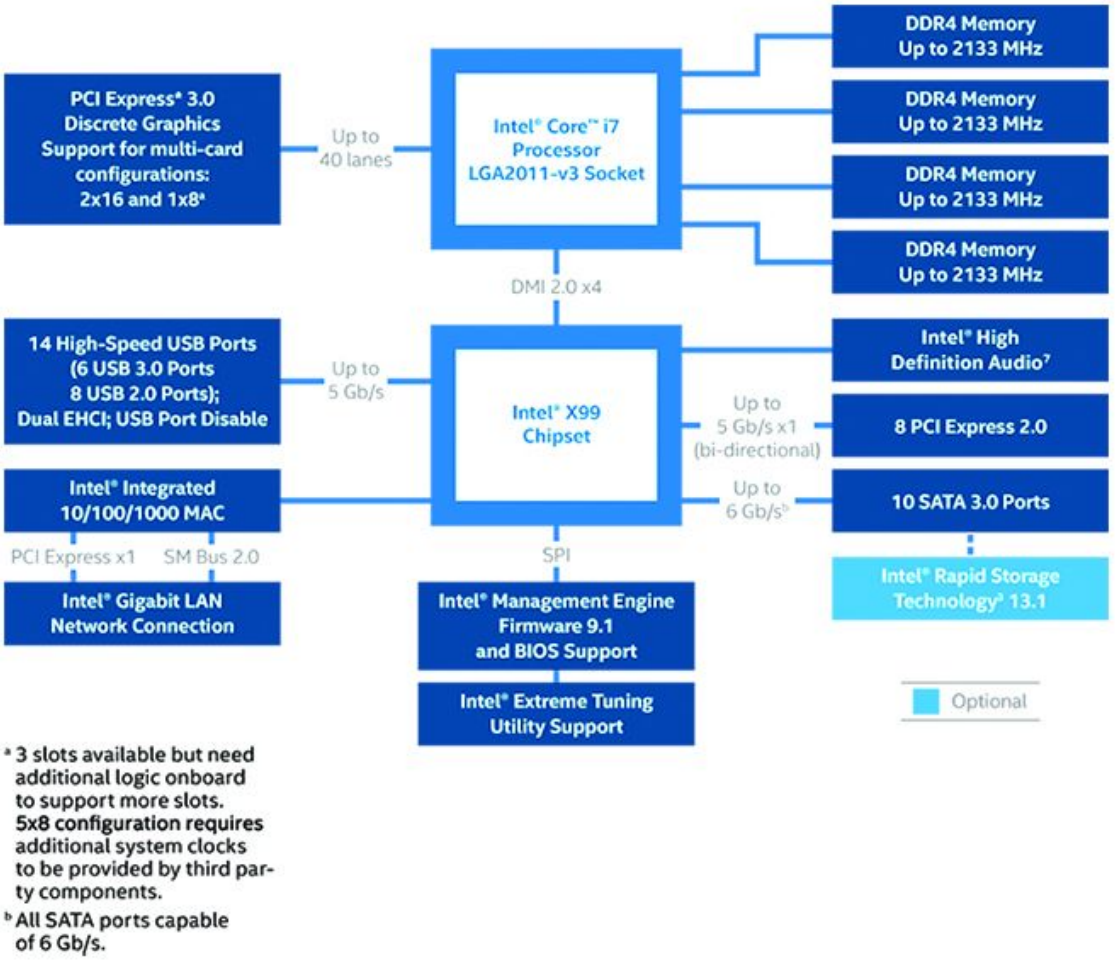
Recuerda: Aunque sea poco conocido, el bus PCI es el más importante del computador.

Los fabricantes de cpu especifican y aclaran qué tipo de conexión cpu-pci proporcionan sus procesadores. Por ejemplo, el siguiente diagrama está sacado de una descripción comercial del procesador Ryzen:



Actualización (2019) :

En algunas cpu modernas, existe un conexionado de buses que evita el pci para algunos dispositivos:



Organización de los dispositivos pci

En un computador puede haber varios buses pci, pero en los ordenadores domésticos y de oficina, normalmente sólo nos encontraremos uno de ellos por cuestiones de **coste**. En este bus habrán conectados diferentes dispositivos, cada uno de ellos ocupa un "slot" o zócalo (aunque físicamente no lo parezca). Cada dispositivo a su vez puede desempeñar diferentes funciones y ser visto como varios dispositivos en uno (por ejemplo una tarjeta de vídeo puede también generar audio por el hdmi y por tanto tendría dos funciones diferentes vistas como dos dispositivos distintos)

En este punto, ya se puede establecer una manera de **identificar dispositivos** en un entorno pci. En el siguiente ejemplo vemos un dispositivo conectado al cuarto slot (empiezan en 0) y hacemos referencia al dispositivo que realiza la segunda función:

Dominio	Bus	Slot	Función
--	00	03	1

Estos cuatro valores son los que identifican un dispositivo pci concreto.

Cad bus pci es capaz de hacer un rastreo y descubrimiento de los dispositivos conectados. El sistema operativo puede, de esta forma, de conocer el hardware conectado. Esto -en teoría- puede hacerse en caliente

Más sobre el reconocimiento de dispositivos pci (hotplugging)

Bus USB

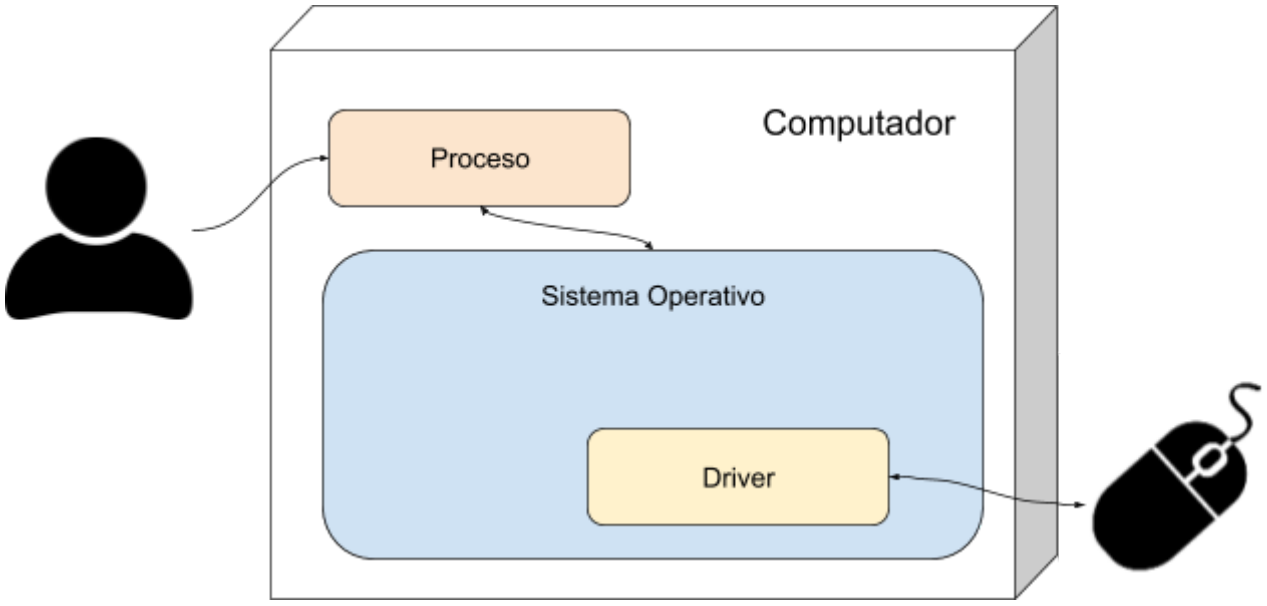
[por hacer]

Manejadores o Drivers en Linux

(recordatorio opcional) ¿Qué es un Driver?

Un driver es la **parte del sistema operativo** que se encarga de activar, configurar, manipular e interactuar con algún dispositivo hardware. Un driver es similar a un proceso o programa pero su única función al ejecutarse es controlar el dispositivo para el que fue concebido. Los programas o procesos de usuario usan el ratón por medio del sistema operativo, y de éste, una parte importante es el driver.

Por ejemplo, para un ratón, el driver es el "proceso del sistema" que lee los movimientos que el ratón detecta y las pulsaciones de los botones. Los ratones son dispositivos muy estándar y los drivers universales pueden usarse con casi cualquiera de ellos. Si algú modelo es muy especial, necesitará un driver específico.

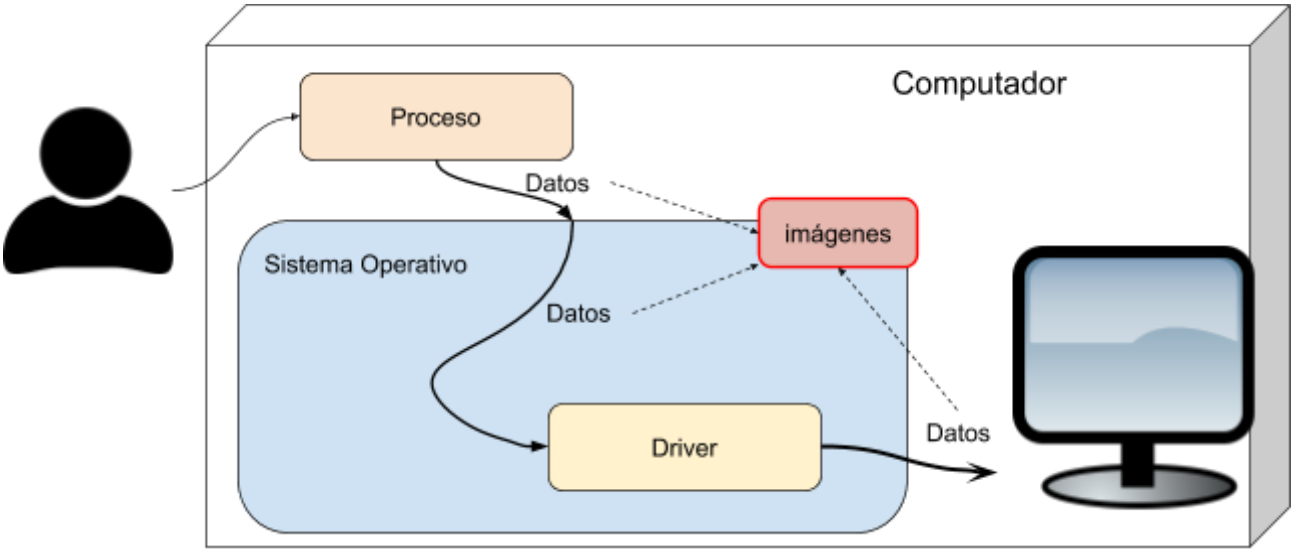


El driver realiza las siguientes acciones:

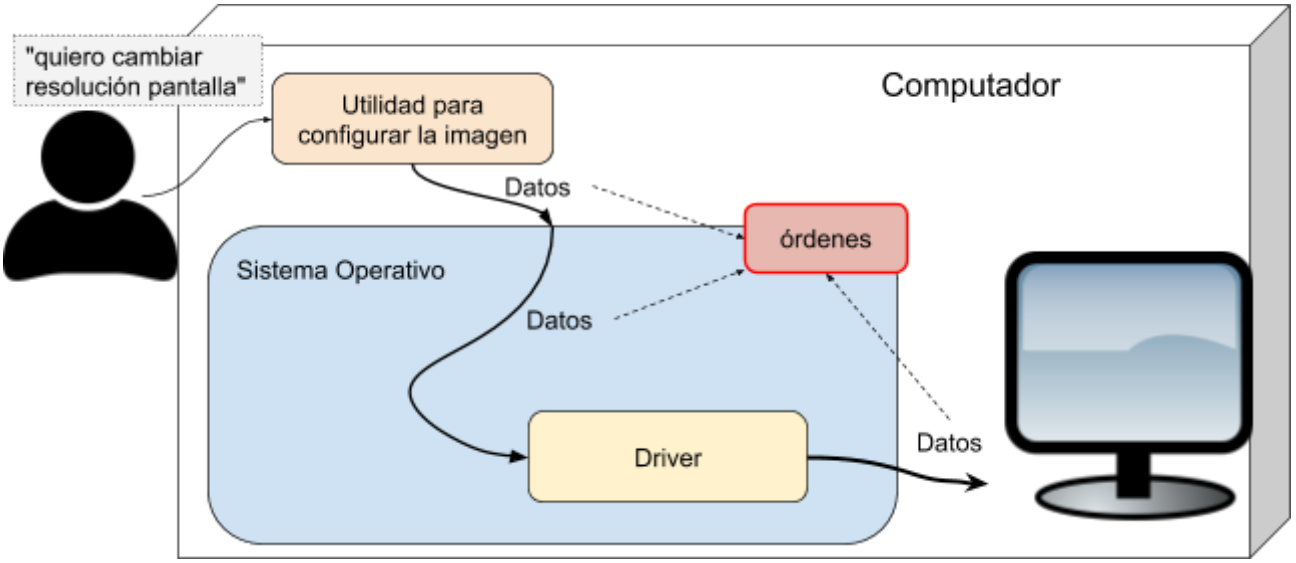
- **Descubre** el estado en el que está el dispositivo e informa al Sistema Operativo. Por ejemplo, descubre el consumo de energía de un ratón usb.

- **Atiende** al dispositivo si éste lanza un aviso especial. Por ejemplo, si se pulsa un botón en el ratón
- **Transmite** datos al dispositivo conectado, estos datos serán:
    - Comandos que activan, apagan o cambian el estado del dispositivo.
    - Datos en sí para que el dispositivo los transmita al exterior.
- **Lee datos** captados por el dispositivo que se pretenden tratar en el computador. Por ejemplo, leer los datos escritos en un disco duro

Un ejemplo más completo: en un computador, todo lo que se dibuja en la pantalla, ha sido transmitido del sistema operativo al driver y del driver al dispositivo. Las imágenes que se visualizan, son datos que terminan materializando una imagen en la pantalla.



Pero en ocasiones se desea cambiar de resolución. Es el driver el que al ejecutarse, se comunica con la tarjeta gráfica y activa el cambio de resolución. En este caso, el driver también se comunica con el dispositivo, pero los datos enviados no son una imagen, sino una orden al dispositivo. El driver y el dispositivo conocen un protocolo de comunicación y conjunto de órdenes.



Los buses también tienen sus propios drivers o manejadores y son clave en la detección del resto de dispositivos conectados.

En el mundo windows, es habitual que el fabricante del dispositivo cree el driver apropiado y lo entregue en un disco o lo ofrezca para descarga desde la web. Para los dispositivos más básicos (ratón , teclado, etc) windows ya incorpora drivers desde la instalación.

Historia breve de los drivers en Linux

Hace muchos años, no había tanta variedad ni tantas novedades y avances en los dispositivos que un computador personal tenía. Básicamente un computador siempre tenía:

- Un **teclado** conectado al puerto ps/2 o DIN
- Un **adaptador gráfico** (EGA/CGA/VGA o Vesa). El monitor no se consideraba periférico como hoy en día, tan sólo un visualizador.
- Un **disco** duro IDE
- Una **disquetera**
- Uno o dos **puertos serie** y uno **paralelo**.
- Un **ratón** conectado al puerto **serie**.
- Una **Impresora** conectada al puerto paralelo anterior
- Un **bus ISA** (el antecesor del pci)

... y ¡ Ya está ! No habían buses pci (y mucho menos usb), ni agp, ni tarjetas gráficas complejas. Hoy, la mayoría de estos dispositivos ha evolucionado (para ser conectado mediante usb/pci) o han desaparecido (bus paralelo o serie, disquetera,dvd, etc.)

Recompilar el núcleo

Ante tal rígido panorama, los desarrolladores de linux incluyeron en el propio núcleo

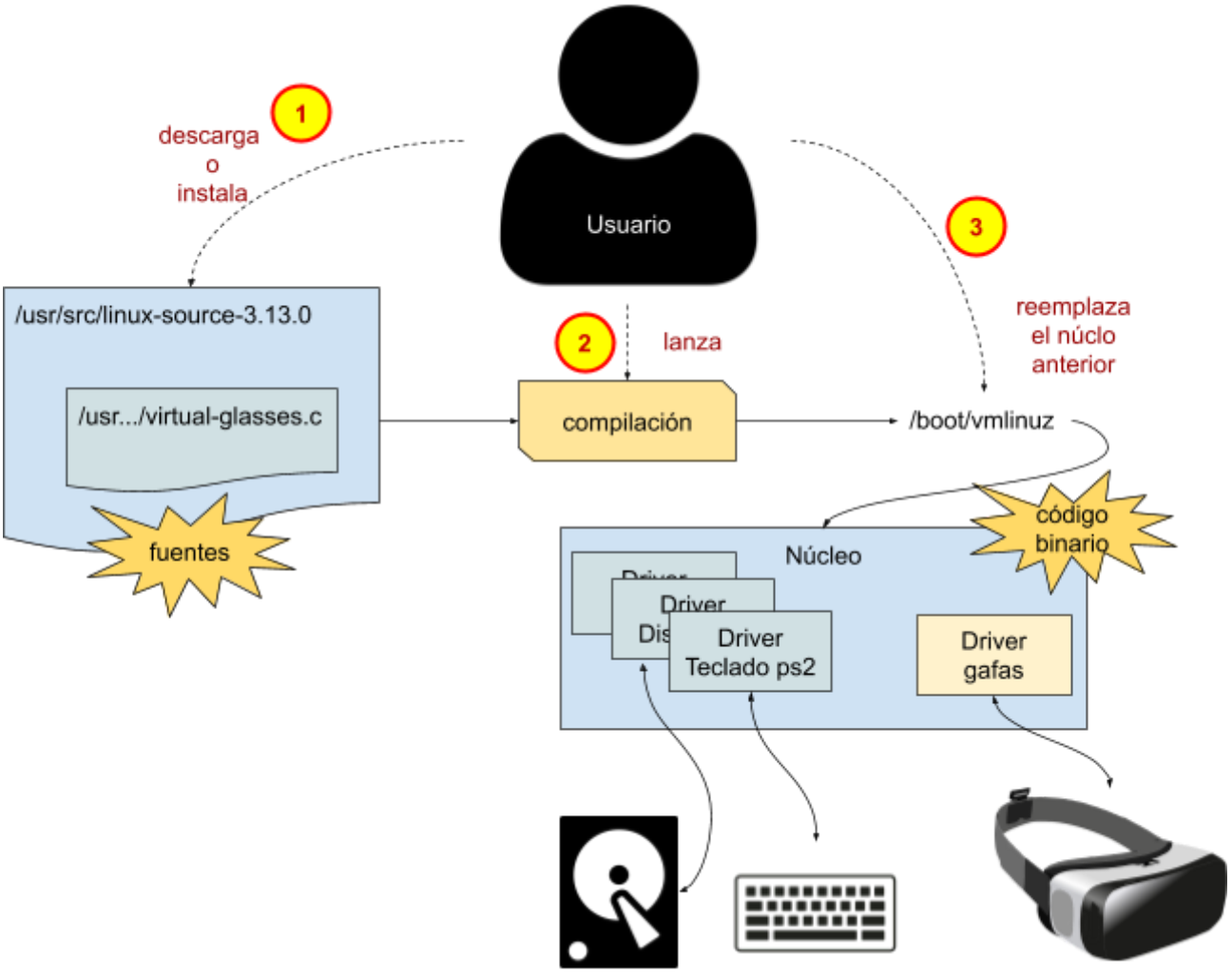
todos los drivers necesarios para usar estos dispositivos.

Cuando el núcleo (fichero `/boot/vmlinuz`) era cargado e inicializado, los drivers estaban allí ya listos para tomar el control de los dispositivos.

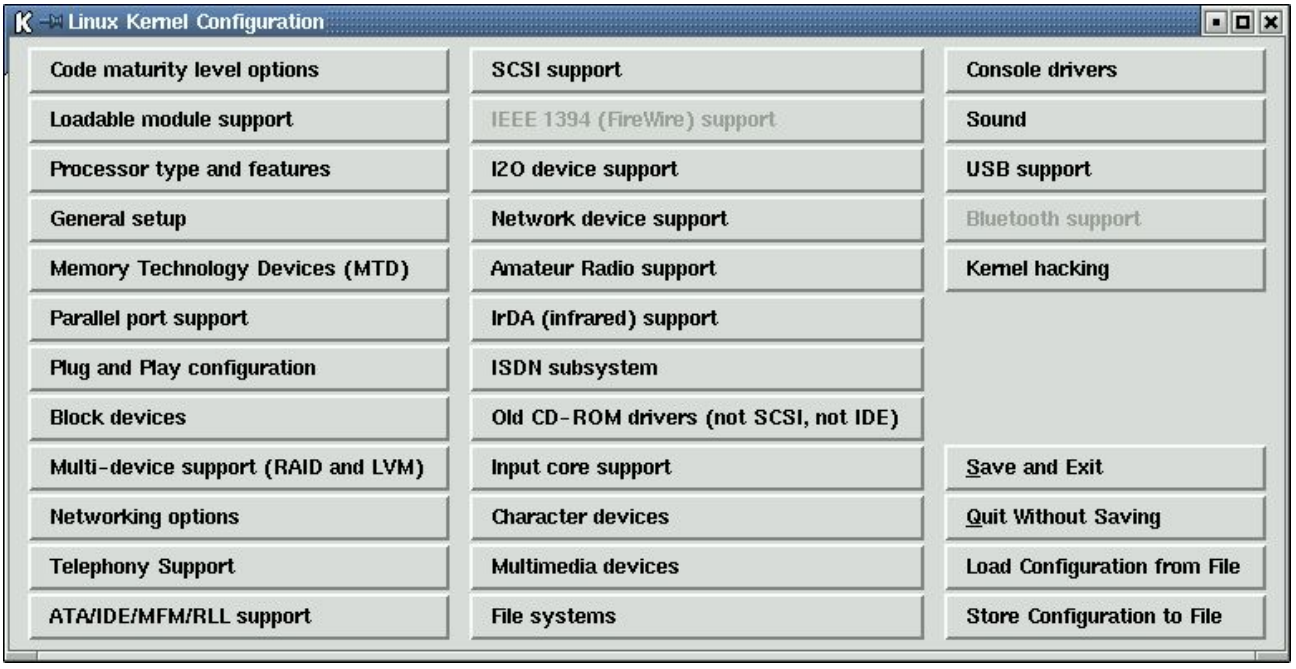
Añadiendo drivers de forma primitiva

Si se deseaba usar algún otro hardware, había que *recompilar* el núcleo incluyendo el código fuente del driver en la compilación. "Recompilar" es generar una imagen ejecutable a partir de los ficheros en código fuente que escriben los programadores. Hoy en día esto normalmente no es necesario durante la vida de un equipo informático. Sin embargo, antaño era habitual que en la instalación de Linux, se dejase instalados los ficheros de código fuente (en lenguaje C y ensamblador), el compilador y todo lo necesario en el sistema para facilitar al usuario esta operación. Como resultado de la compilación, el usuario obtenía un nuevo núcleo de sistema operativo en forma de fichero (vmlinuz). Éste se instalaba y, a partir de ese momento, ya se podía iniciar el computador usando este nuevo núcleo con los drivers añadidos para el nuevo dispositivo. Comparativamente, hoy en día un servicio de actualización descarga e instala actualizaciones que incluyen nuevos núcleos. Eran los

tiempos en los que era cierto que "*no todo el mundo puede usar linux, has de ser mu listo!*"



A la hora de compilar, existía una utilidad para permitir al usuario elegir los drivers a incluir en un paso previo a la propia compilación. Por ejemplo, en la siguiente pantalla se elegía qué drivers compilar dentro del núcleo mismo. Lo que ves es el menú principal, cada sección estaba repleta de dispositivos a incluir.



Más información sobre la elección de drivers en
http://how-to.wikia.com/wiki/How_to_configure_the_Linux_kernel

...Pero "*hace años que llegó el futuro*",

- Multitud de fabricantes sacaban hardware novedoso.
- Los buses se actualizaban por otros nuevos, muy capaces y más rápidos, listos para conectar muchos más dispositivos.
- El hardware clásico evolucionaba y se complicaba tanto, que requería drivers nuevos y muy ajustados como los de las tarjetas gráficas (o muchos otros: los ratones incorporaban botones nuevos y ruedas, los teclados luces y teclas programables).
- La placa base añadía sensores de temperatura y voltaje (que cuenta como otro periférico), la propia cpu era vista como un dispositivo con su driver (suena raro), algunas soluciones a problemas nuevos pasaban por tener algo parecido a drivers que no controlaban ningún dispositivo. etc. etc. etc.

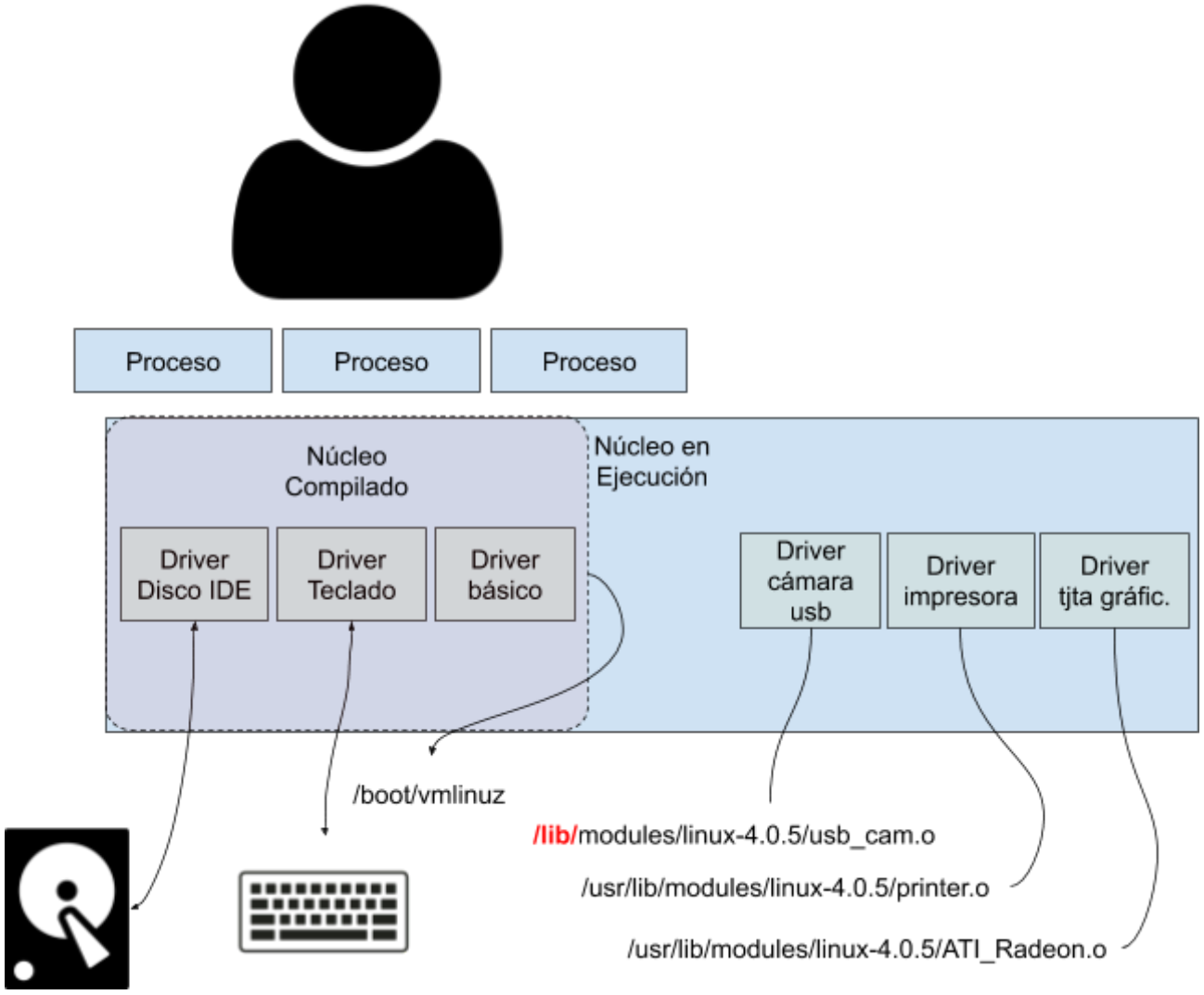
Además, empezó a ser habitual ampliar con más hardware una máquina que ya estaba instalada y mucho de este hardware se añadía en caliente, mientras el sistema está en funcionamiento.

Cada nuevo dispositivo o bus requería su driver y el número de estos desbordaba la conveniencia de "meterlos" en núcleo de linux.

Módulos

Llegado un momento, se hizo necesario abandonar la rigidez y monstruosidad de meter todos los drivers en el núcleo. Y se desarrolló la capacidad de cargar "módulos", que

simplemente **son ficheros de código ejecutable que pueden ser cargados** y pasan a formar parte del núcleo . Esta carga se hace durante el funcionamiento normal del computador .(la misma idea que las bibliotecas dinámicas). En general, un módulo es un driver (aunque existen módulos con otras utilidades)



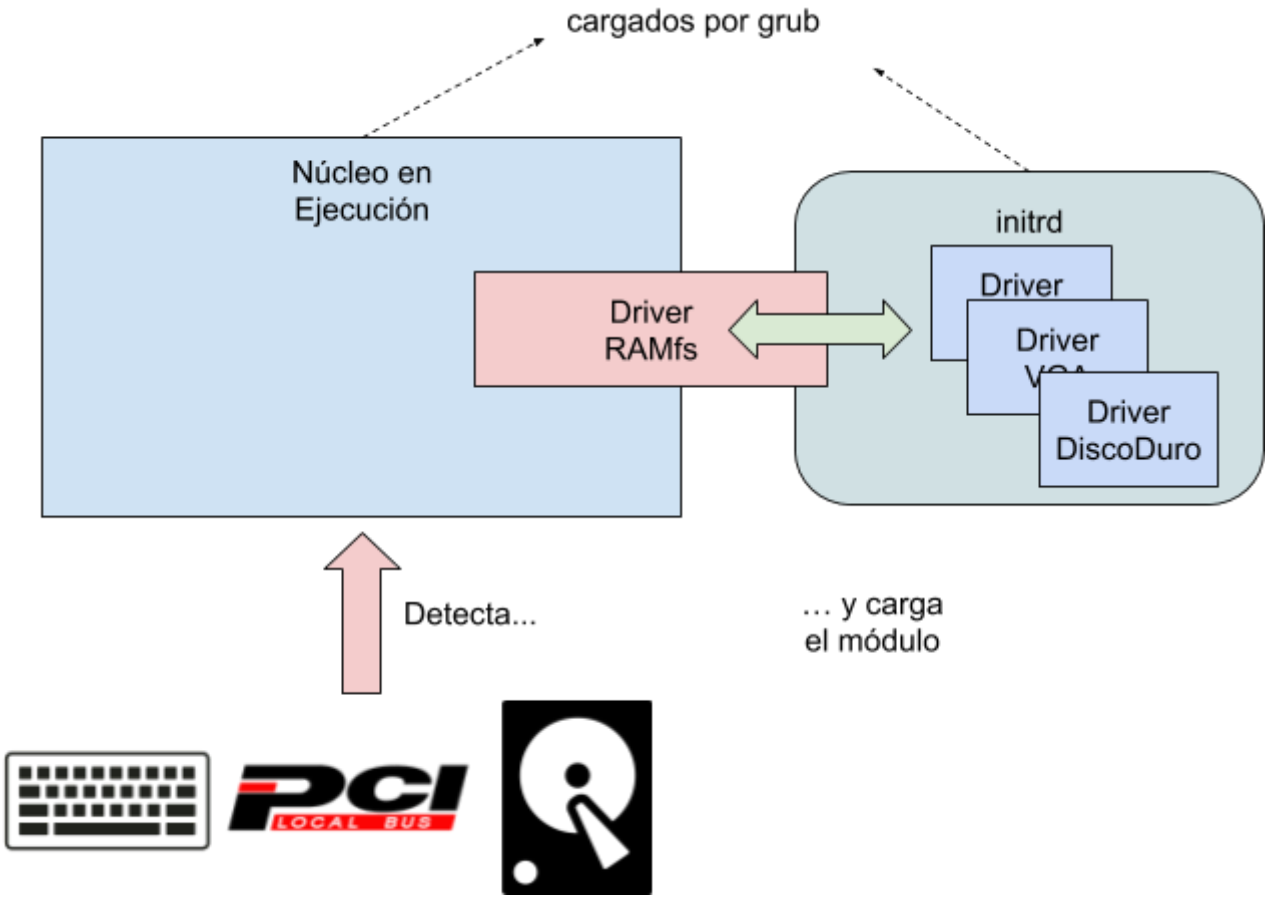
Salvo los dispositivos más básicos y necesarios para el arranque, todo el resto de drivers se implementó como módulos. Hoy en día, los drivers son módulos que se cargan después de que el núcleo esté en funcionamiento. Puedes verificar qué módulos hay cargados en tu sistema con el comando "lsmod"

Arranque

Con el tiempo, esta aproximación de concebir todos los drivers como módulos ha ido extremándose hasta el punto de que durante el arranque se carga una imagen virtual de un sistema de archivos en RAM. En ese sistema de ficheros se incluyen módulos para todo el hardware posible que pueda ser necesario durante el arranque (discos scsi, raid, arranque por red, arranque con "splash screen", etc.). El núcleo así ya no necesita incorporar internamente ningún driver excepto el necesario para acceder a ese sistema de ficheros en RAM. De esta

forma, **el núcleo arranca de la siguiente forma:**

1. Se inicia el ordenador y Grub (el gestor de arranque principal) carga el núcleo en memoria **vmlinuz**. Esto es el sistema operativo en sí, ya podría ejecutarse, pero ¡No tiene drivers para acceder al disco duro y seguir cargando cosas!
2. Seguidamente, grub carga una imagen de sistema de archivos virtual en memoria. En ese sistema de archivos existen módulos (drivers) de muchos dispositivos. Este sistema de archivos es cargado a partir de un archivo llamado **initrd.img**.
3. El núcleo cargado en el punto 1 se inicia y accede a este sistema de archivos virtual en memoria.
4. El núcleo, gracias a esta imagen virtual de drivers entre los que están incluidos drivers de buses pci y usb, **detecta el hardware real** y carga los módulos adecuados desde el sistema de ficheros virtual. Se puede decir, que los módulos se cargan "de la ram a la ram" porque todo está en la ram todavía en este punto.
5. Los drivers incorporados al núcleo van ejecutándose e inicializando los dispositivos reales, entre ellos el disco duro y el sistema de ficheros físico primario (el que se montará en "/")
6. reflejarla importancia del parámetro root en el descubrimiento de la partición de arranque. "linux está ciego yconfía en grub.info"
7. Cuando todo el hardware está reconocido y listo, el núcleo cambia el sistema de archivos principal (montado en "/") del virtual al que existe en alguna partición de los discos detectados.
8. El núcleo puede acceder ya al disco duro, donde están los módulos en forma de ficheros y puede seguir cargando más módulos y/o reiniciar el hardware necesario.



Esta imagen virtual de drivers es lo que se conoce como "initrd"

Realiza el ejercicio de reconocimiento de los ficheros de arranque que está al final del documento

¿Cómo se distribuyen y dónde residen los módulos?

Los módulos se distribuyen principalmente en las imágenes "ISO" que instalan un sistema Linux. La habitual instalación de linux, deja los módulos copiados en un directorio concreto. Durante la vida de un sistema, es posible que se actualice el núcleo y con él se deben actualizar o preparar los módulos para ese núcleo nuevo.

Por ello, para el caso de tener una instalación con más de un núcleo, se decidió que tener un directorio por cada núcleo (aunque se puedan repetir los ficheros en ambos) y el nombre del directorio es por tanto

```
/lib/modules/VERSION_DEL_NUCLEO/kernel/
```

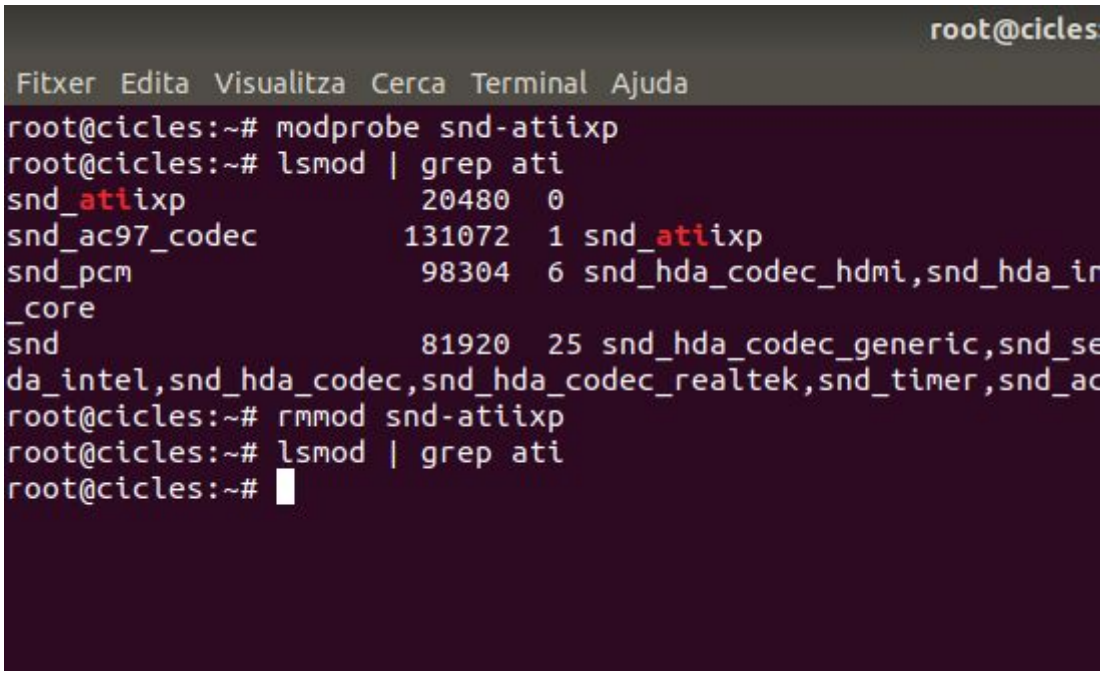
(en ubuntu)

Dentro de este directorio hay todo un árbol de directorios donde están organizados todos los drivers de la distribución linux.

Carga, verificación y descarga de módulos

- `modprobe` : cargar un módulo.
- `lsmod` : listar módulos cargados
- `rmmod`: eliminar un módulo cargado

En la siguiente captura se inserta un módulo (driver), se muestra que está cargado, se descarga y se muestra que ya no está.



Abre un terminal y repite el proceso de la captura para familiarizarte con los comandos anteriores

¿Cómo se detecta el hardware?

Detectar el hardware significa que el sistema operativo sabe que hay tal o cual dispositivo en el computador. A raíz de ello, se cargará o ejecutará el driver asociado a ese dispositivo y el usuario podrá finalmente hacer uso de él.

Un poco de historia. Detección en sistemas antiguos

Asegurar que el Sistema Operativo sea conocedor del todo el hardware que tiene el computador siempre ha supuesto un problema. Hace años, cuando un usuario añadía un hardware al computador, nada especial ocurría de por sí. El computador seguía funcionando igual y el hardware no era ni detectado, ni se instalaban automáticamente los drivers.

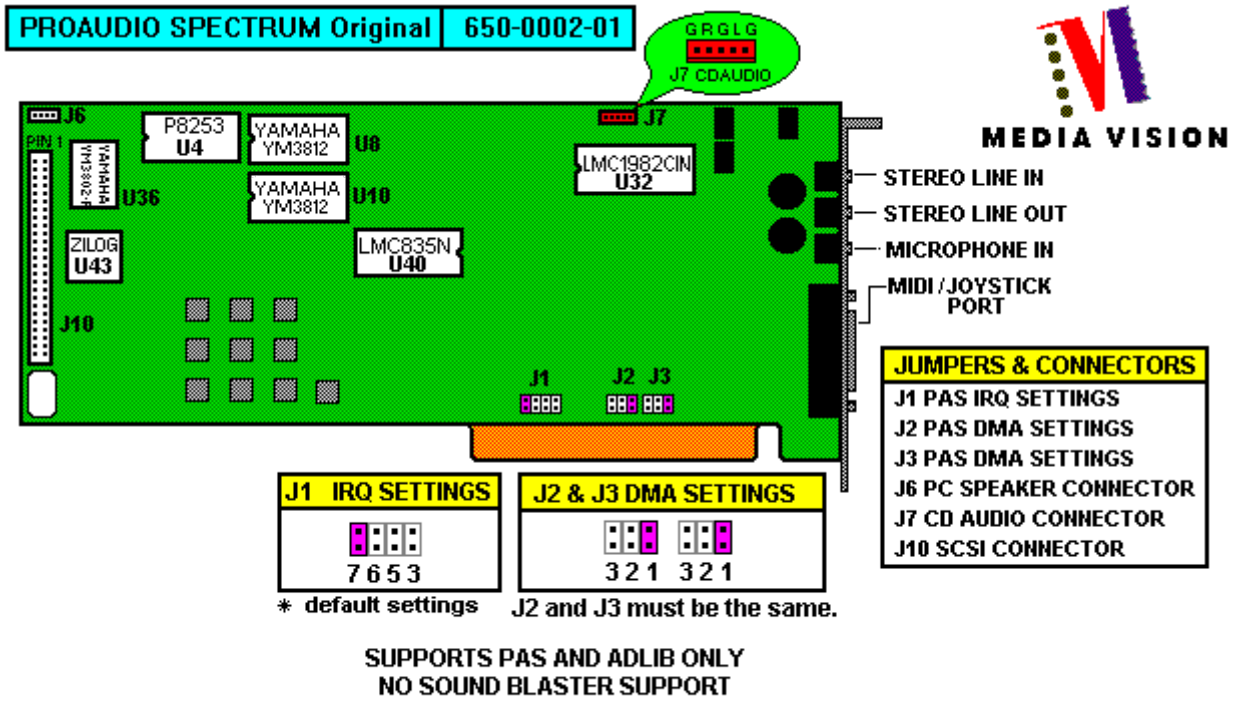
Por ejemplo, añadir una tarjeta de sonido no hacía ni que el sistema operativo la

detectase, ni que las aplicaciones que emitían audio, efectivamente, sacasen sonido.

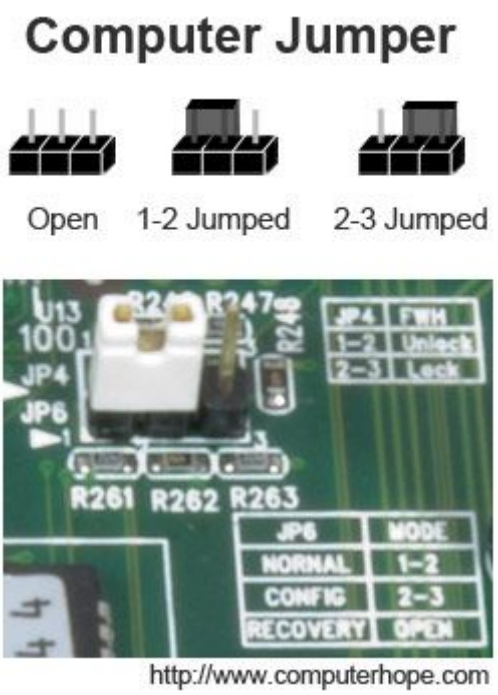
Para usar el nuevo dispositivo había que instalar el driver. Y éste ejecutaba una rutina de instrucciones que trataba de detectar la presencia del hardware. En ocasiones, el propio usuario tenía que indicar algunos parámetros de muy bajo nivel para ayudar al driver a detectar con éxito el hardware. En los tiempos de MS-DOS, había incluso juegos que incorporaban su propio driver para tarjetas de sonido.

> En el ejemplo de la tarjeta de sonido, era frecuente que se pudiese elegir un número de interrupción y dirección de E/S cambiando algunos contactos ¡ En la propia tarjeta! Después, al instalar el driver o arrancar el juego, se le tenía que indicar qué parámetros habías elegido y el driver quedaba configurado con ellos.

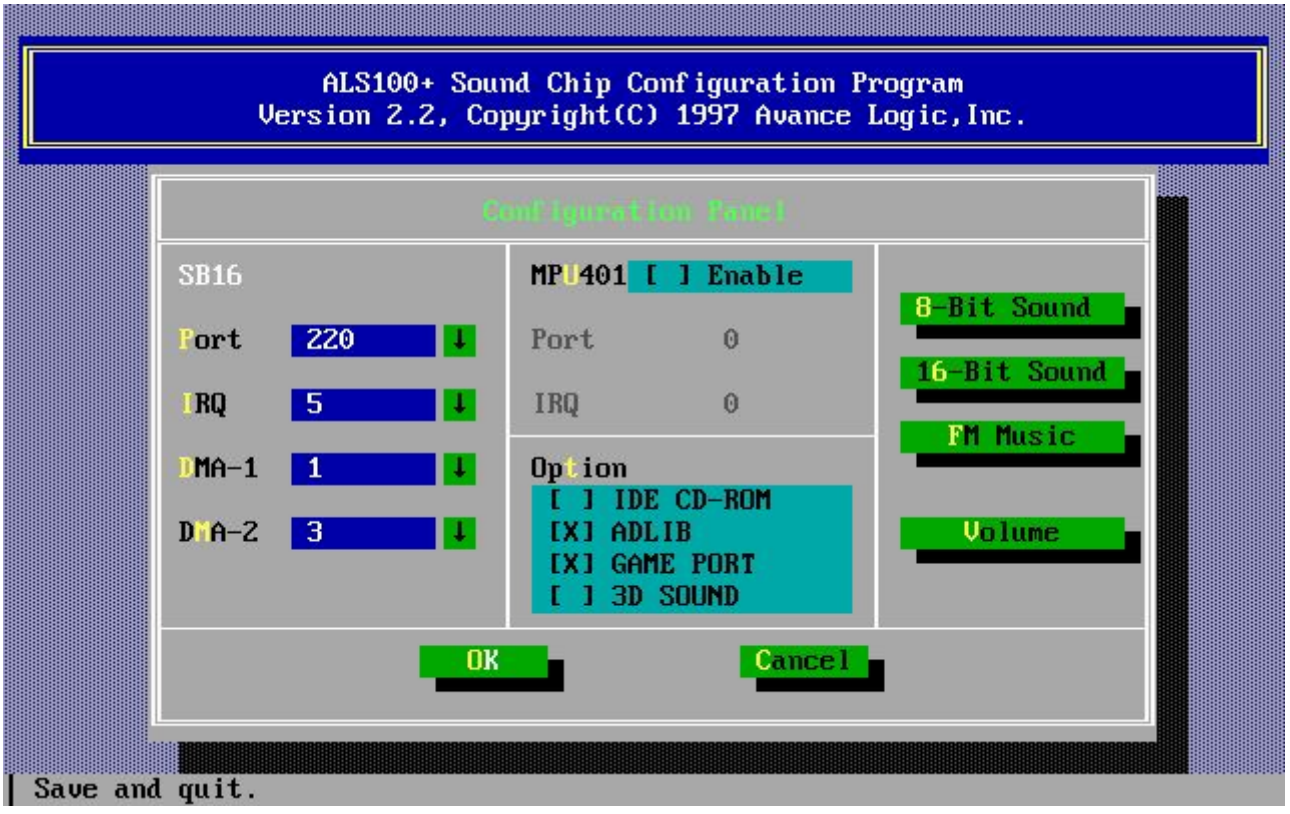
La siguiente imagen está sacada de las instrucciones de una tarjeta de sonido. En ella, se indica la ubicación física de los jumpers (contactos) de selección de canales DMA y número de Interrupción:



Este es el aspecto de unos jumpers:



A continuación se puede ver la pantalla de configuración que la utilidad de instalación de una tarjeta de sonido usaba para pedir al usuario la configuración hardware especificada mediante jumpers

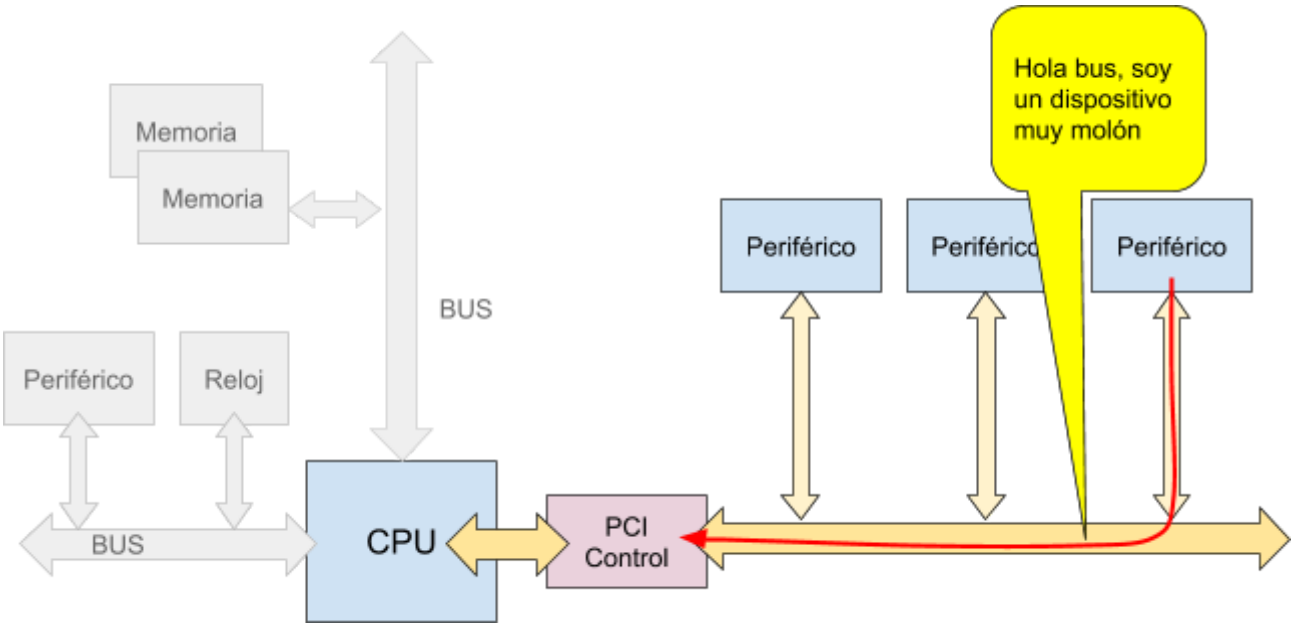


Observa que hay que "configurar el driver" indicando el número de puerto entrada/salida, el número de interrupción, y el canal DMA... cosas muy de hardware.

Estándar plug & play

Más o menos al tiempo que Windows 95, se introdujo un estándar para los fabricantes de dispositivos y computadores que permitiese simplificar la instalación y configuración del hardware. En general se exigía que las placas base y el firmware (bios) fuesen más "inteligentes" y que los dispositivos "hablasen" al computador y se "anunciasen" a éste. Así, el computador detecta los dispositivos porque ellos informan al bus de su presencia. Este estándar se llamó "Plug And Play" (pnp)

Aunque así sigue haciendo falta instalar el driver, se consigue una mínima configuración y estabilidad sin esfuerzo por parte del usuario. El sistema plug & play terminó de funcionar totalmente con el bus PCI, que incorpora esta característica de forma nativa.



<poner imagen de placa base recibiendo la información de una tarjeta gráfica>

La detección del hardware significa que el bus donde va conectado el dispositivo pueda establecer un mínimo diálogo con el mismo y descubra aspectos básicos del dispositivo. Este descubrimiento se reporta después al Sistema Operativo y finalmente al usuario (todo ello sin implicar a los drivers del dispositivo, pero sí a los del bus)

Recuerda: "Plug & Play" no significa que lo enchufes y funcione, tan sólo que lo enchufes y sea reconocido mínimamente para instalar los drivers con más facilidad o usar unos ya existentes.

Propiedades del hardware

Detectar el hardware consiste en que el computador, gracias al bus, descubra que tiene conectado algo y después, averigüe todo lo posible de ese "algo". Casi todo el hardware tiene unas propiedades comunes.

- **Fabricante** ("vendor" en inglés)
- **Modelo** ("device")
- **Tipo** o clase de dispositivo
- **Bus** o clase de bus en el que se conecta

Dado que estas propiedades básicas y universales son estándar, los dispositivos, computadores, buses y sistemas operativos son diseñados de forma que, sin necesidad de drivers, al insertar un dispositivo, éste comunica las propiedades al sistema operativo. Y estos parámetros constituyen la detección del hardware. (ver sección modalias a continuación)

A partir de aquí, cada dispositivo tiene unas propiedades particulares (tipo de chasis, consumo, canales de audio, número de botones etc.). Sin embargo, estas propiedades, al ser específicas de algunos tipos o modelos de hardware, no pueden ser descubiertas de forma tan sencilla e inmediata por el bus. Hay que instalar el driver adecuado del dispositivo y al inicializarlo, se puede acceder a todos los detalles del dispositivo gracia al driver

Recuerda: Los dispositivos plug & play tienen 4 propiedades comunes: Fabricante, modelo, tipo y bus al que se conecta.

Recuerda: Una cosa es la identificación de un dispositivo para el sistema operativo (bus, conexión, función) y otra la información que se obtiene de él para saber sus características (fabricante, modelo, tipo y bus)

La norma (pci??) indica que todo dispositivo debe ser capaz de informar de las 4 propiedades citadas, el bus debe ser capaz de escuchar esa información y el SO debe ser capaz de recogerlo.

Detección en sistemas modernos.

Para comprender totalmente cómo funciona la detección y carga de drivers en un sistema linux vamos a tener que conocer antes algunos conceptos básicos y aparentemente desconectados. ¡Ánimo!

A partir de ahora, aparecerá el directorio /sys en muchos momentos de la explicación. Más adelante se tratará con más profundidad su cometido. De momento el directorio /sys contiene una estructura de directorios virtuales que refleja las características y jerarquía de todo el hardware existente en el ordenador.

En la identificación del hardware detectado, el modalias es un elemento muy importante.

Modalias

Un "modalias" es una línea de información que describe un dispositivo concreto y que aglutina toda la información sobre el tipo de dispositivo e identificación del mismo. (sólo de identificación, no describe todos los detalles). Es como la marca y modelo de cualquier artículo. Por ejemplo (en el ordenador donde se redacta este documento), aquí tienes varios modalias:

(en fichero `/sys/devices/pci0000:00/0000:00:1d.0`)

```
pci:v00008086d0000A298sv00001043sd00008694bc06sc04i00
```

(en fichero `/sys/devices/pci0000:00/0000:00:13.2/usb1/1-0:1.0/modalias`)

```
usb:v1D6Bp0002d0400dc09dsc00dp00ic09isc00ip00in00
```

(en fichero: `/sys/devices/virtual/dmi/id/modalias`)

```
dmi:bvnFUJITSUSIEMENS:bvr1.01C:bd06/20/06:svnFUJITSUSIEMENS:pnAM
ILOPa1510:pvrNotApplicable:rvnFUJITSUSIEMENS:rnAMILOPa1510:rvrNo
tApplicable:cvnFUJITSUSIEMENS:ct10:cvrNotApplicable:
```

(`/sys/devices/system/cpu/modalias`)

```
cpu:type:x86,ven0002fam000Fmod0048:feature:,0000,0001,0002,0003,
0004,0005,0006,0007,0008,0009,000B,000C,000D,000E,000F,0010,0011
,0013,0017,0018,0019,001A,001C,0020,0021,0022,0023,0024,0025,002
6,0027,0028,0029,002B,002C,002D,002E,002F,0030,0031,0034,0036,00
37,0038,0039,003B,003D,003E,003F,0064,0070,0071,0075,007A,0080,0
08D,00C0,00C1,00C2,00C3,00C4,010F
```

Recuerda: Un "modalias" es una línea de información que describe un dispositivo concreto

Formato del modalias

Cada línea de modalias tiene un formato e información ligeramente diferente en función del subsistema (usb, pci, placa base, cpu, etc.). Veámos un ejemplo para entender el formato para el caso pci:

(`/sys/devices/pci0000:00/0000:00:01.0/0000:01:05.0/modalias`)

```
pci:v00001002d00005975sv00001734sd000010B8bc03sc00i00
```

Las primeras letras antes de los dos puntos (":"), indican el subsistema y dependiendo de él, el resto define otros parámetros.

- subsistema: "pci" . Se indica a qué parte del computador pertenece este dispositivo.

Esta **primera palabra** determina la información que se espera en el resto de línea.

- `v00001002`
  - v : indica que le sigue un número con el fabricante ("vendor")
  - 00001002: código numérico que identifica al fabricante.

- `d00005975`
  - d: dispositivo o "modelo"
  - 00005975: número o código de modelo

- "sv" y "sd" son "subvendor" y "subdevice" respectivamente. Identifican pequeñas diferencias entre ejemplares del mismo modelo.

- `bc03sc00`
  - bc : "Base class" . Aquí se indica el tipo de dispositivo en cuestión.
  - sc: "Subclass"

La línea modalias de una cpu tiene otro formato y es más extensa:

```
cpu:type:x86,ven0002fam000Fmod0048:feature:,0000,0001,0002,0003,0004,0005,0006,0007,0008,0009,000B,000C,000D,000E,000F,0010,0011,0013,0017,0018,0019,001A,001C,0020,0021,0022,0023,0024,0025,0026,0027,0028,0029,002B,002C,002D,002E,002F,0030,0031,0034,0036,0037,0038,0039,003B,003D,003E,003F,0064,0070,0071,0075,007A,0080,008D,00C0,00C1,00C2,00C3,00C4,010F
```

En el caso de la cpu, se debe indicar qué funciones tiene o no tiene esa cpu directamente en el propio modalias (los procesadores no tienen driver estrictamente, pero se identifican como un dispositivo)

Interpretación de los datos de un modalias

Los datos vistos en un modalias están codificados (cada fabricante tiene un código). En principio no podemos saber fácilmente los detalles de un dispositivo. Hay que recurrir a comandos o a mirar en bases de datos en los que encontrar más información a partir de los códigos que nos da el dispositivo con el modalias.

Esta web tiene el listado de fabricantes y modelos:

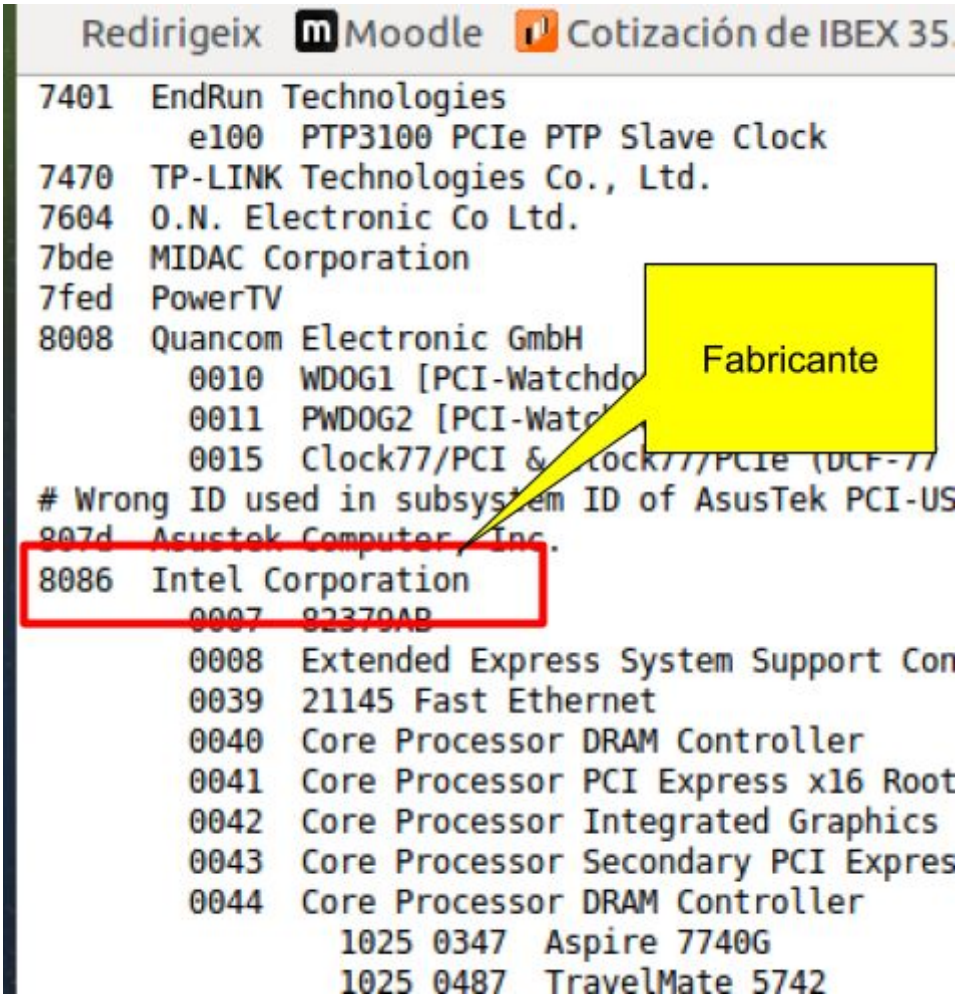
http://pci-ids.ucw.cz/

y concretamente el listado que se mantiene actualizado con los datos conocidos:

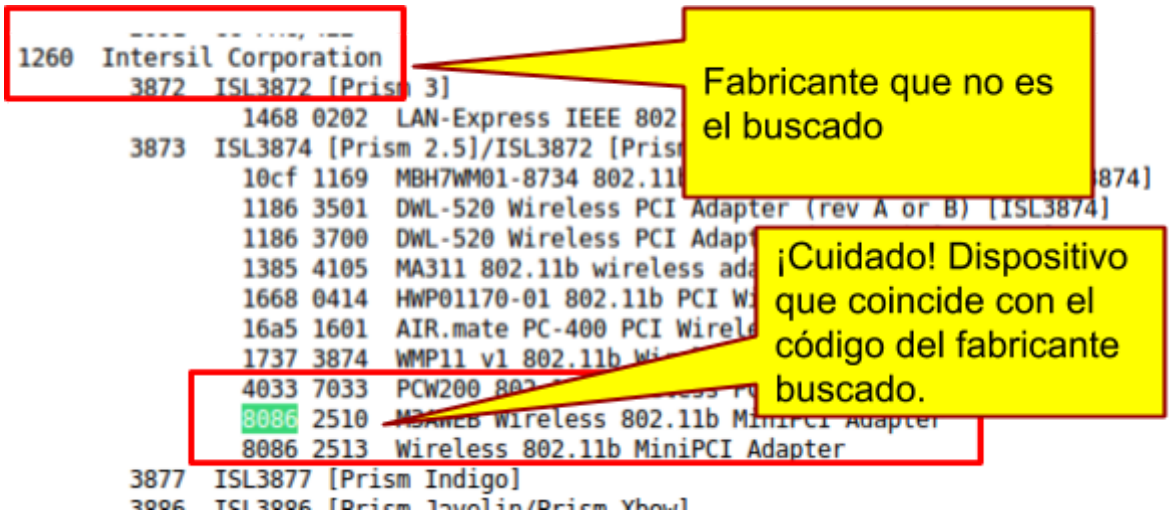
http://pci-ids.ucw.cz/v2.2/pci.ids

Procedimiento:

1. Se busca el código del vendedor, que aparece en la columna exterior (a la izqda del todo). Ojo porque el formato de la página no es muy amigable y hay claves en posiciones que no son las del vendedor pero lo aparentan. .
    a. por ejemplo, si buscas 8086 como fabricante deberías encontrar esto:



    b. pero no hacer caso de esto



2. Después se busca el dispositivo dentro de la lista del fabricante. ¡ Ojo con no salirte a

la lista de otro fabricante! ... porque cada fabricante puede tener un listado muy extenso que hace que te pierdas en la página.

El anterior modalias

(/sys/devices/pci0000:00/0000:00:01.0/0000:01:05.0/modalias)
pci:v00001002d00005975sv00001734sd000010B8bc03sc00i00

corresponde con el siguiente dispositivo

Fabricante: `AMD`
Clase de Dispositivo: "`vga compatible controller`"
Modelo: "`RS482M [Mobility Radeon Xpress 200]`"
Subvendor: `Fujitsu Technology Solutions`

Recuerda: cada modelo de dispositivo del mundo mundial tiene su modalias particular.

Un modalias identifica a un dispositivo concreto (no a un ejemplar, sino a un modelo. Esto es, no hay número de serie implicado en principio). Esto será útil para que el sistema operativo seleccione el driver adecuado.

Carga de drivers

< Extraído de aquí. >

Describamos la secuencia de eventos importantes que ocurren **al insertar un dispositivo** en caliente con el computador ya en marcha.

1. Supongamos que el núcleo de linux ha arrancado, está en ejecución y puede examinar los buses.
2. En algún bus, se realiza una búsqueda hardware y se detecta un dispositivo recién conectado.
3. El driver del bus consulta la identificación del dispositivo, se obtiene su modalias y se publica en `/sys`.
4. **Algo mágico pasa aquí**
5. Se carga el driver o módulo adecuado para gestionar dicho dispositivo.
6. El dispositivo funciona.

El pungo 4 encierra un misterio. Hemos de hacernos dos preguntas clave

1. ¿Cómo sabe Linux cuál de todos los módulos es el adecuado para el dispositivo detectado?

2. ¿Quién promueve la carga del driver? ¿Qué parte del núcleo o qué proceso se activa para activar la carga del driver?

Encontrar el driver adecuado.

Esto es "complifácil", empecemos por el final.... :

El módulo o driver sí que sabe qué dispositivos son los suyos. Lo lleva escrito bien adentro del fichero, en su código y cuando sea cargado y puesto en marcha, verificará que el dispositivo es efectivamente el que él puede gestionar.

Puedes preguntar a un driver o módulo por los dispositivos que él es capaz de gestionar con el comando `modinfo` . En la siguiente captura de pantalla se observa parte de la información que muestra el comando sobre un driver llamado 8255_pci:

```
root@ordenadorcillo:~# modinfo 8255_pci
filename:
/lib/modules/4.18.0-10-generic/kernel/drivers/staging/comedi/drivers/8255_pci.ko
license:
...
alias:          pci:v00001093d00001800sv*sd*bc*sc*i*
alias:          pci:v00001093d000017D0sv*sd*bc*sc*i*
alias:          pci:v00001093d00001250sv*sd*bc*sc*i*
...
depends:        comedi_pci,comedi_8255,comedi
...
```

Las líneas marcadas con "`alias:`" especifican un patrón de modalias que son gestionables por este driver. El patrón es una forma compacta de escribir a un conjunto de modalias. Por ejemplo, la línea

```
alias:      pci:v00001093d000017D0sv*sd*bc*sc*i*
```

Describe a todos los dispositivos conectables al pci con vendedor `00001093` dispositivo `000017D0` y para cualquier subvendor (sv*), subdispositivo (sd*), etc. Los asteriscos son caracteres comodines que pueden reemplazar a cualquier cadena... es decir "que ahí va lo que sea!"

Pero linux no puede ir examinando o cargando uno por uno todos los módulos y esperando que uno de ellos "acepte" gestionar el dispositivo recién detectado. Sería un tiempo inmenso e intolerable.

Para acelerar esto, Hay una base de datos que **relaciona dispositivos y drivers**. Esa base de datos tiene la descripción en forma de modalias de todos los dispositivos para los que se tiene un driver, y para cada uno de ellos, el módulo o driver asociado. Y la identificación del dispositivo en esta base de datos es el "modalias". Cuando el núcleo

descubre los datos de identificación del dispositivo (esto lo hace el driver del bus), se forma su modalias y a continuación se consulta en el fichero que es la base de datos. Este fichero, en ubuntu se encuentra en:

```
/usr/lib/modules/4.9.31-ubuntu/modules.alias
/lib/modules/4.2.0-16-generic/modules.alias
```

(donde, evidentemente, "`4.9.31-ubuntu`" es mi versión de núcleo y mi directorio particular)

Fichero modules.alias

Así pues, **el fichero "modules.alias" es una base de datos que relaciona dispositivos y drivers.** A continuación se muestran algunas líneas presentes en este fichero recogidas de varias partes de él y comentadas

(este primer ejemplo lleva una cabecera añadida para diferenciar partes)

```
#     --------------- modalias ------------------- --- driver ---

alias cpu:type:x86,ven0000fam0006mod005C:feature:* intel_rapl_perf
alias cpu:type:x86,ven0000fam0006mod0055:feature:* intel_rapl_perf
alias cpu:type:x86,ven0000fam0006mod005E:feature:* intel_rapl_perf
alias cpu:type:x86,ven0000fam0006mod004E:feature:* intel_rapl_perf
```

En estas líneas...

- a la cpu de tipo x86 (pc normales)
- del vendedor intel (0000f)
- de la familia 6 variantes 005C,0055,005E, etc.
- y que tengan cualquier característica,

  .... se le asigna el driver intel_rapl_perf.

Observa que se pueden usar caracteres comodín como el asterisco ("*"). Ello indica que en ese campo del modalias puede ir cualquier cosa. El efecto de estas líneas, es que si durante el procedimiento de detección de hardware se detecta una cpu de este fabricante,modelo, tipo, etc., se cargará el driver "`intel_rapl_perf`" que aparece en esa línea.

Otro ejemplo:

```
alias cpu:type:x86,ven*fam*mod*:feature:*00C2* kvm_amd
```

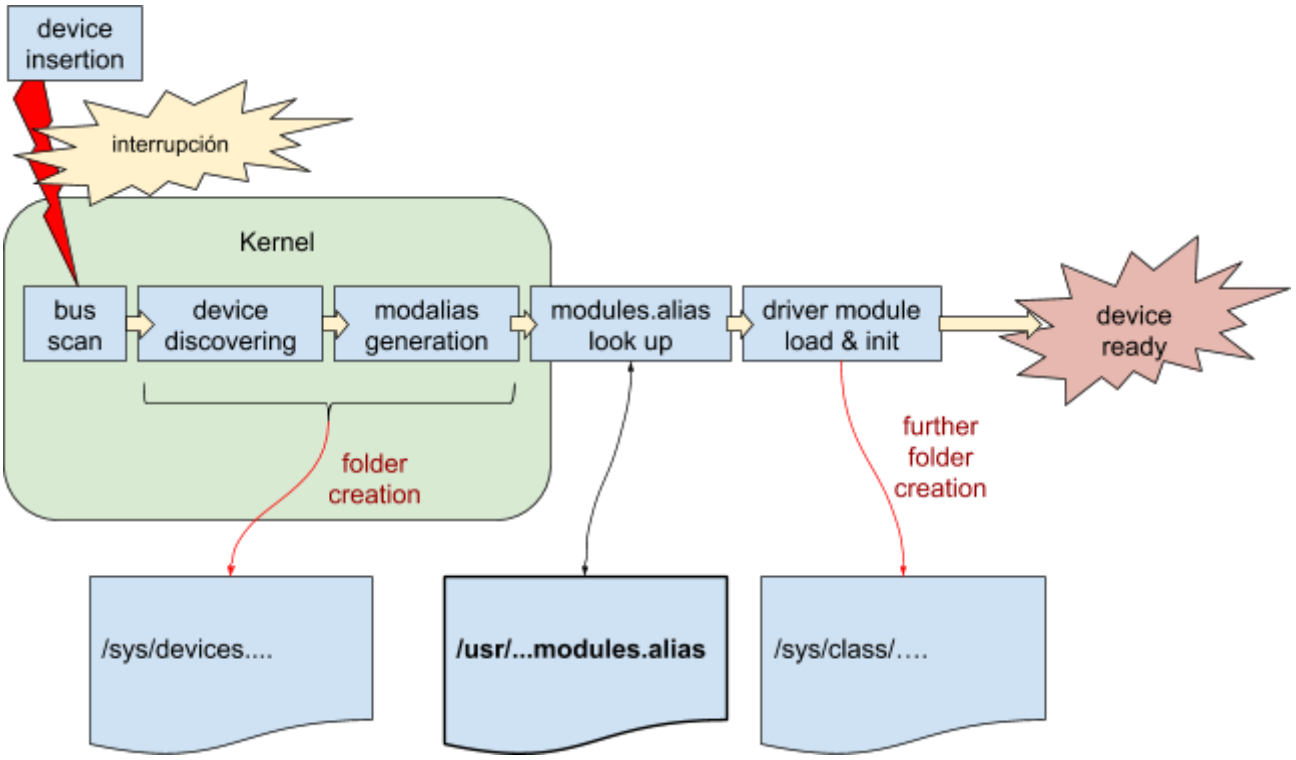
Aquí, se asigna el módulo kvm_amd a cualquier cpu de tipo x86, de cualquier fabricante, cualquier variante y familia que contenga cualquier característica adicional a la 00C2 (que por

supuesto, ahora no tengo ni puta idea de lo que puede ser, pero diría que es algo de la virtualización). Este kvm_amd no es estrictamente un driver, pero sí que es un módulo que será cargado asociado a una característica de la cpu, que la propia cpu sabe que tiene e informa.

En esta otra línea del fichero:

```
alias pci:v00001002d000067CFsv*sd*bc*sc*i* amdgpu
```

Se asigna, cualquier dispositivo del vendedor 00001002 de tipo de dispositivo 000067CF es tratado con el módulo `amdgpu` (¿será una tarjeta gráfica radeon?)



Creación del fichero modules.alias

Este fichero no es estático, sino que ¡ puede cambiar !

Como hemos dicho antes, cada módulo sabe qué dispositivos puede manejar y se le puede preguntar. Una utilidad llamada depmod se encarga de preguntar uno por uno a todos los módulos acerca de los dispositivos que puede manipular, con esa información re-genera el fichero modules.alias.

Evidentemente, esto tiene sentido hacerlo:

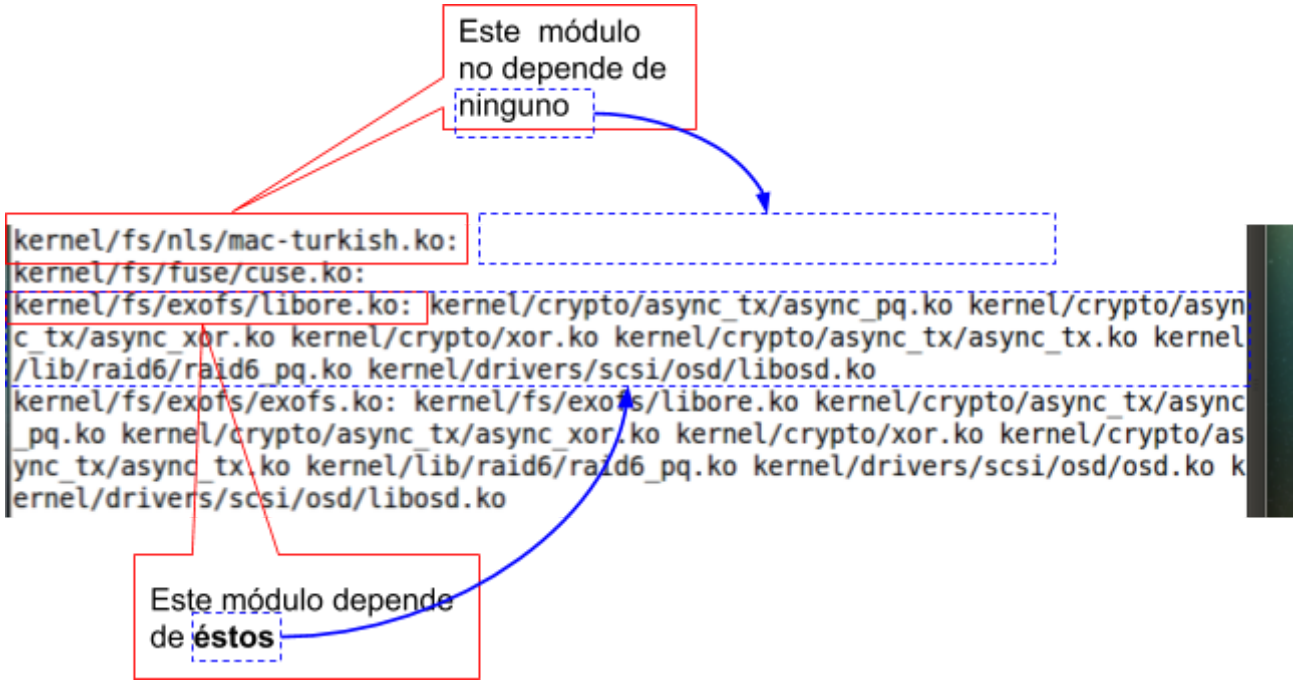
- Al actualizar la versión del núcleo o elemento básico
- Al añadir un módulo nuevo.
- Al instalar el sistema.

- Cada 3 de septiembre de año bisiesto si el administrador a las 10:02 ha bostezado.

Dependencias entre módulos

Algunos módulos requieren la presencia de otros para poder funcionar. Estas dependencias están escritas en un fichero llamado "`modules.dep`" ubicado en el mismo directorio que el fichero "modules.alias" (`/lib/modules/version/modules.dep`) y obtenido de forma similar al anterior modules.alias.

Cuando se va a insertar un módulo, se comprueban la dependencias y se realiza una carga de módulos previa, que pueden a su vez, tener otras dependencias. El fichero de dependencias también lo genera el comando depmod. A continuación puedes ver un extracto del fichero



¿Quién promueve la carga del driver de un dispositivo?

El procedimiento y los pasos ya están aclarados, volvámoslos a citar de todas formas:

1. Supongamos que el núcleo de linux arranca y puede examinar los buses.
2. En algún bus, se realiza una búsqueda y se detecta un dispositivo no gestionado todavía.
3. El driver del bus consulta la identificación del dispositivo. Con ella, se obtiene su modalias y se publica en `/sys`.
4. **Se carga el driver o módulo adecuado para gestionar dicho dispositivo**.
5. El dispositivo funciona.

Este paso 4 es más complejo de lo que parece. La pregunta es :

¿Qué es lo que inicia el punto 4 anterior? ¿Es el núcleo o un proceso?

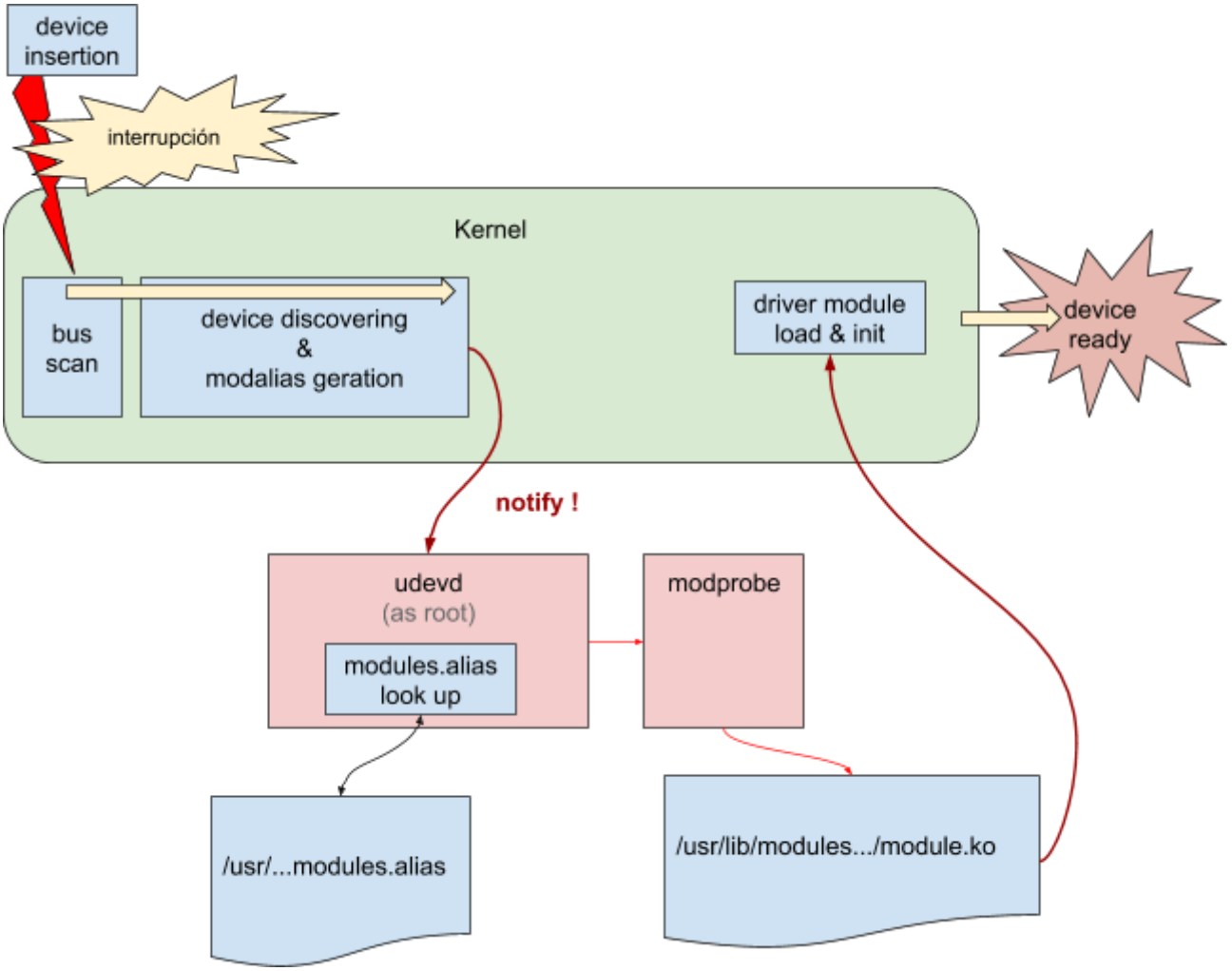
El modalias se obtiene fácilmente gracias al driver del bus, pero resulta que ¡ El núcleo se desentiende de buscar el driver adecuado y cargarlo ! (en versiones antiguas sí lo hacía, ahora ya no)

El núcleo se limita a informar a cualquier proceso que quiera darse por enterado de que hay un hardware nuevo. Dicho de otra forma, el núcleo espera que algún proceso sea el responsable de buscar el driver y desencadenar la carga del módulo. Evidentemente estamos hablando de algún proceso o servicio que se ejecuta con permisos de root.

Esta delegación de trabajo del módulo a algún proceso tiene sentido por:

- Hay que mirar el fichero modules.alias, modules.dep, etc. Rara vez el núcleo ha de leer ficheros por sí mismo
- El núcleo no puede iniciar una operación incierta, no se sabe si existe módulo o driver para el dispositivo detectado.
- El núcleo no puede esperar. Si algo va a tardar, que sea otra cosa la que espere... como un proceso.
- Existen restricciones de seguridad que pueden aplicarse. Las debe haber configurado el administrador y el núcleo no tiene por qué estar pendientes de ellas.
- Existe hardware que puede tener algún trato especial (por ejemplo para montar siempre un disco externo usb en la misma ubicación)

Por todo ello, se ha creado un paquete llamado **udev** ("udevd es el nombre del proceso) que se encarga de realizar todo este trabajo, que se resume en cargar los drivers adecuados (no es realmente cargar, sino "solicitar que se cargue")



The uevent message contains information about the device (example). This information contains registered vendor and model identification for devices connected to buses such as PCI and USB.

1. Udev parses these events and constructs a fixed-form module name which it passes to modprobe

2. modprobe looks under /lib/modules/VERSION for a file called depmod.alias which is generated when the kernel is installed and that maps the fixed-form module names to actual driver module file names. See Are driver modules loaded and unloaded automatically? for more details about the process — that answer describes the earlier days when the kernel called modprobe directly, but the way modprobe and module aliases work hasn't changed.

   See also Michael Opdenacker's presentation "Hotplugging with udev" which has more examples and describes other aspects of device management with udev, and the Linux from scratch guide which has a section on how the fixed-form module names are defined.

3. modprobe loads a module by calling the init_module system call. It doesn't interact

with sysfs in any way.

4. When the module is loaded, the kernel creates an entry for it in /sys/module. Any entry that appears elsewhere in sysfs is up to the code in the module (e.g. a module with a driver for a type of USB devices will call some generic USB support code that adds an entry under /sys/bus/usb/drivers).

http://www.linuxfromscratch.org/lfs/view/development/chapter07/udev.html :

7.3.2.3. Module Loading

Device drivers compiled as modules may have aliases built into them. Aliases are visible in the output of the modinfo program and are usually related to the bus-specific identifiers of devices supported by a module. For example, the snd-fm801 driver supports PCI devices with vendor ID 0x1319 and device ID 0x0801, and has an alias of "`pci:v00001319d00000801sv*sd*bc04sc01i*`". For most devices, the bus driver exports the alias of the driver that would handle the device via sysfs. E.g., the `/sys/bus/pci/devices/0000:00:0d.0/modalias` file might contain the string "`pci:v00001319d00000801sv00001319sd00001319bc04sc01i00`". The default rules provided with Udev will cause udevd to call out to /sbin/modprobe with the contents of the MODALIAS uevent environment variable (which should be the same as the contents of the modalias file in sysfs), thus loading all modules whose aliases match this string after wildcard expansion.

In this example, this means that, in addition to snd-fm801, the obsolete (and unwanted) forte driver will be loaded if it is available. See below for ways in which the loading of unwanted drivers can be prevented.

The kernel itself is also able to load modules for network protocols, filesystems and NLS support on demand.

https://doc.opensuse.org/documentation/leap/reference/html/book.opensuse.reference/cha.udev.html#sec.udev.drivers

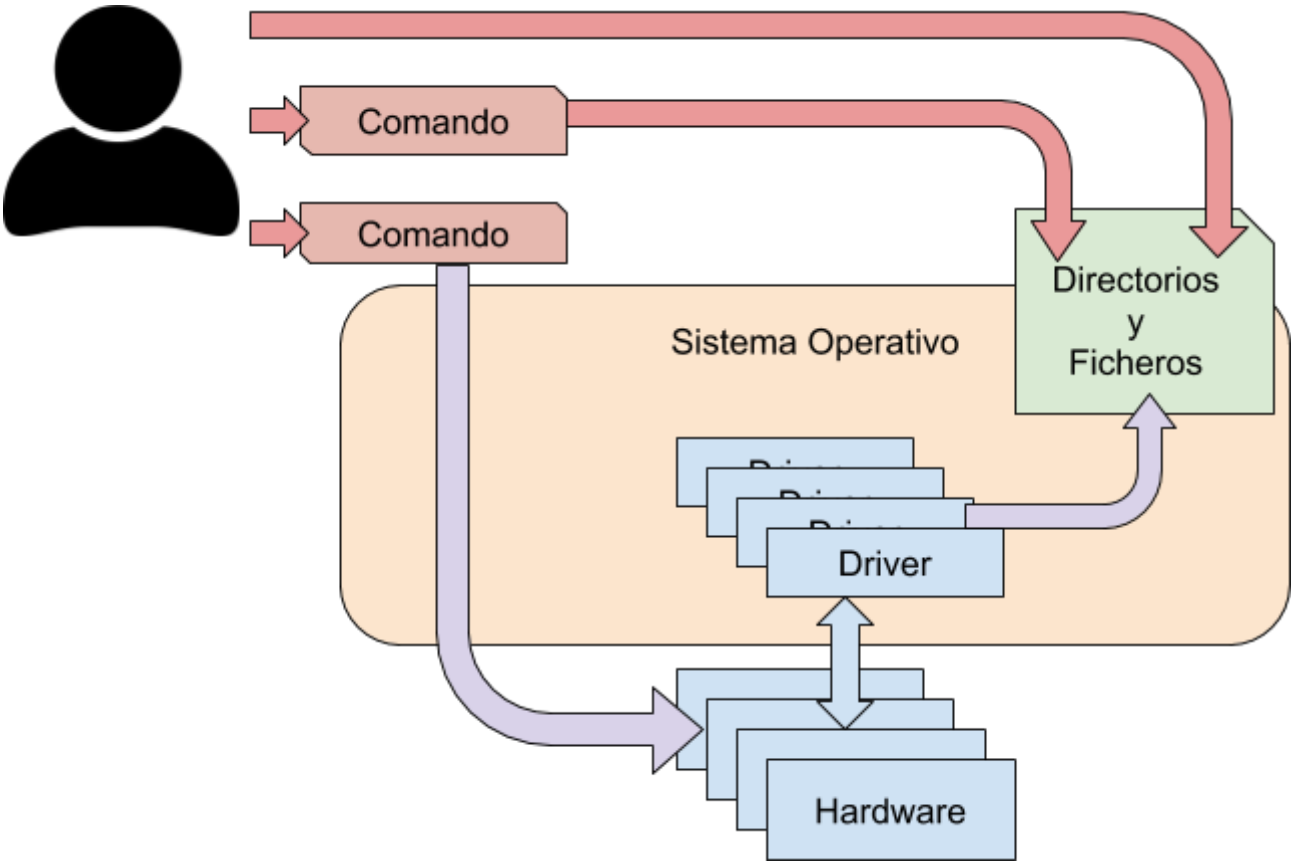
https://stackoverflow.com/questions/25175960/which-drivers-are-used-by-usb-mouse-in-linux-kernel?rq=1

¿cómo asignar un driver específico a un hardware?

Información del hardware del computador "/sys"

Cuando los dispositivos están ya detectados y su driver está activo y en funcionamiento, el usuario puede desear saber acerca del hardware de su computador. Dispone de dos maneras de averiguar los detalles del hardware del computador:

1. **Consultar directorios** y ficheros donde el núcleo expone la información del hardware conocido
2. **Ejecutar comandos** que muestran información sobre el dispositivo. Estos comandos a su vez pueden acceder a los directorios del punto anterior o directamente al hardware



Directorio /sys

< ¡Atención! Estas secciones sobre /sys son durillas por tostón (no por difíciles) y quizá podrían obviarse. Pero si lo haces, habrán bastantes ejercicios que no podrás hacer y/o no entenderás completamente, incluyendo algunos bastante interesantes. De hecho, Este documento se inició como un intento de entender esa infernal carpeta>

El núcleo de Linux ofrece **información del hardware a través de un conjunto de ficheros virtuales que cuelgan de /sys** (ficheros que no existen en el disco duro realmente pero aparecen en el árbol de directorios y parecen ficheros normales). Al leer alguno de estos ficheros se visualiza información concreta del hardware de la máquina (modalias,

temperatura, versión del dispositivo, configuración de algún parámetro, etc.) Pero esa información se saca del propio núcleo o incluso del dispositivo en el mismo momento de la lectura del fichero.

https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

Estos ficheros están en el directorio /sys, que es un directorio virtual también. Conforme el núcleo (durante el arranque y después) va descubriendo dispositivos e instalando drivers, nuevos directorios y ficheros se van creando dentro de /sys para ir reflejando todo el hardware encontrado.

Dicho de otra forma: El núcleo informa de las características del hardware "inventándose" ficheros y carpetas que cuelgan del directorio /sys

Elementos del directorio /sys

https://lwn.net/Articles/646617/

El directorio /sys es una representación de las estructuras de datos del núcleo que describen el hardware de la máquina. Si un dispositivo es reconocido por el núcleo, automáticamente se crea un directorio y, dentro de él, ficheros asociados en algún lugar dentro de /sys.

Por tanto hay al menos un directorio para cada dispositivo y este directorio tiene ficheros cada uno de los cuales tiene información sobre algún parámetro o característica del dispositivo. Leer y, -en ocasiones- escribir en dichos ficheros puede usarse para averiguar o establecer alguna configuración sobre el dispositivo.

A continuación puedes ver un directorio con ficheros y subdirectorios (con más ficheros) creados por el núcleo para un disco duro ATA en la máquina de ejemplo.

```
[arch-fijo ~]# cd  /sys/dev/block/11\:0
[arch-fijo 11:0]# ls
alignment_offset  device            ext_range  queue       slaves
badblocks         discard_alignment holders    range       stat
bdi               events            inflight   removable   subsystem
capability        events_async      integrity  ro          trace
dev               events_poll_msecs power      size        uevent
[arch-fijo 11:0]#
```

Cada uno de esos ficheros contiene una información concreta sobre el disco duro.

Esta **jerarquía de directorios es compleja** y muchas rutas internas están relacionadas entre sí. Para cada dispositivo, puede haber realmente varios elementos en /sys, y en varias subcarpetas diferentes. Hay que tener en cuenta que un dispositivo está conectado en algún bus y por tanto, para empezar, hay varias entidades asociadas al dispositivo:

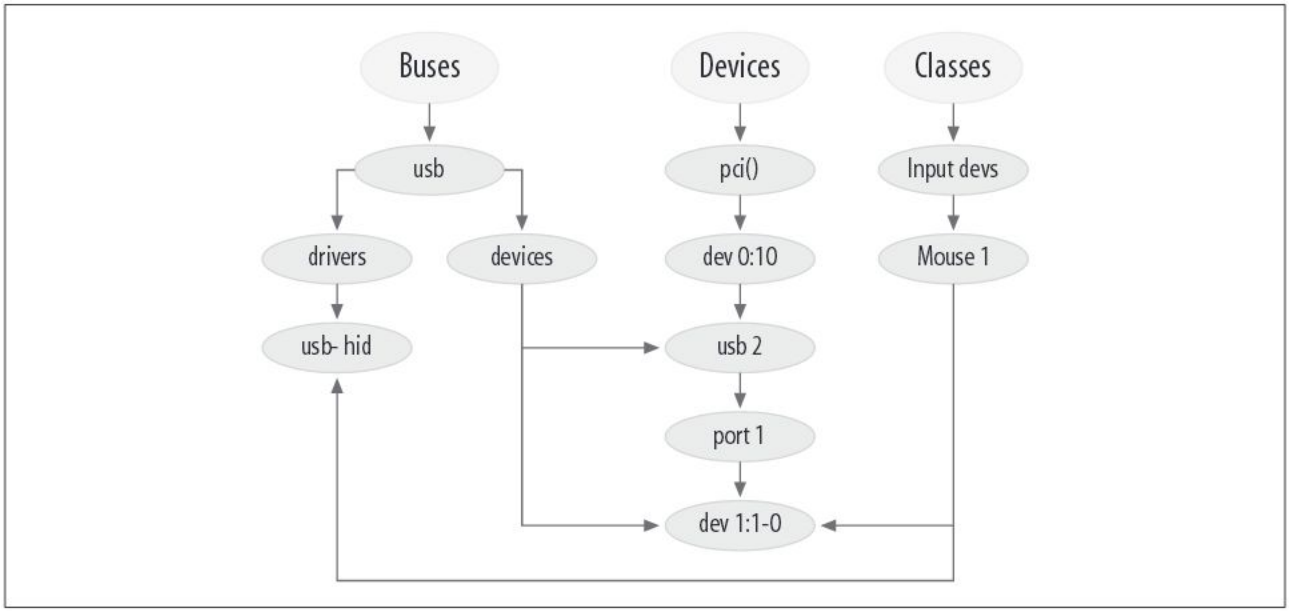
- Dispositivo genérico conectado en algún punto del bus que se ha identificado pero del

cual los detalles de funcionamiento no interesan. Sólo interesa la conexión en sí. Para ello aparece un directorio

Si el dispositivo está conectado a un bus y éste está conectado a otro, aparecerán dos conexiones y, por tanto, dos directorios

- Dispositivo real, gestionado por el driver, que realiza alguna función concreta y tiene varias características propias.
- Driver que está implicado en la gestión del dispositivo. (sí, los drivers también tienen su carpeta dentro de /sys)

Por ejemplo, al insertar un ratón USB, el núcleo y el driver del bus USB detectarán por sí mismos un nuevo "dispositivo USB" y lo darán de alta como tal en algún lugar de /sys, pero al cargar el driver adecuado, éste dará de alta un nuevo dispositivo de tipo "ratón". Ambos dispositivos son el mismo, vistos como "algo USB conectado al computador" y "ratón". El bus USB en sí mismo es un dispositivo conectado al bus PCI y por ello se puede ver una compleja relación de buses, drivers y objetos ¡ Sólo por un ratón !



En el diagrama anterior constituye el conjunto de entradas para un ratón. El dispositivo aparece múltiples veces como

- Un dispositivo conectado a un puerto 1 de un bus "usb 2" conectado a un zócalo del bus pci
- Un dispositivo usb conectado a un puerto concreto usb
- Dentro de la clase de dispositivo "input Dev" (dispositivos de entrada) aparece el ratón como tal

Además

- El bus usb tiene un driver asociado (entre otros) por culpa de este ratón y ello también se refleja en el directorio

La jerarquía de carpetas y subcarpetas reproduce -en algunas partes del árbol-, la jerarquía misma de dispositivos y buses, y cómo están conectados físicamente. Así, dentro de la carpeta del bus pci, encontraremos la carpeta del puente pci-usb, y dentro la de un dispositivo hub usb, y dentro la de algún dispositivo conectado al bus (como el ratón de antes)

ejercicio, dibuja el diagrama hw para tu ratón.

Recuerda: El directorio /sys es una representación de las estructuras de datos del núcleo que describen el hardware de la máquina

El directorio /sys NO se puede usar para los siguientes fines:

- Escribir o leer de un dispositivo (para eso hay otra carpeta distinta a /sys)
- Cargar o descargar algún driver (aunque la información sobre qué driver puede buscarse)
- Activar o desactivar los dispositivos.

Nota: El directorio /sys tiene otros usos adicionales no relacionados con los dispositivos o periféricos.

Jerarquía superior de /sys

Documentación de referencia.

La organización de la carpeta /sys es realmente compleja. El número y niveles de anidación de las carpetas es alto, así como los múltiples enlaces que se hacen desde unos directorios a otros, entrelazándose unos con otros. Es desaconsejable pelear para abarcarlo todo. Sin embargo, el nivel superior (lo que se observa en la propia carpeta /sys) es explicable y al menos, despeja muchas incógnitas acerca de dónde está cada cosa.

Se pueden observar las siguientes carpetas con su contenido:

block	
bus	Cada carpeta representa a un bus (y dentro de cada una, sus dispositivos)
class	Organiza los dispositivos por clases (dos teclados, uno usb y otro ps2 estarán dentro de la misma subcarpeta porque son dispositivos de clase "teclado")
dev	dentro hay dos subcarpetas /char y /block que organizan los dispositivos como antiguamente hacía linux: en dispositivos de bloques (se puede escribir bloque a bloque) y carácter (se escribe un carácter cada vez). Y dentro de cada carpeta aparecen enlaces a dispositivos. Estas carpetas tienen una

	nomeclatura que refleja también una forma clásica de organizar los dispositivos : número mayor y número menor.
devices	Contiene un sistema de ficheros que representa el árbol de dispositivos tal como lo mantiene el núcleo. Se trata de una jerarquía de dispositivos (incluyendo buses)
firmware	Contiene una representación de los dispositivos básicos de la placa base, firmware, gestión de energía, mapas de memoria.
module	Por cada módulo cargado se crea aquí una carpeta con información detallada con el módulo.
fs	Contiene una representación de los sistemas de ficheros en el sistema. Aparecen directorios como "ext4" y "cgroup", entre otros

A continuación tienes unos pocos ejemplos de ficheros.

Directorio (dentro de /sys)	Contenido
`block/sda`	Los datos sobre el disco duro se pueden encontrar en varios ficheros existentes en
`devices/cpu/type`	El tipo de cpu se localiza en el fichero
`devices/system/cpu/cpu0/cpufreq/bios_limit`	La frecuencia máxima de trabajo Establecida en la bios de un core en una cpu dual core se obtiene a partir del fichero:
`devices/virtual/dmi/id`	La información sobre la placa base
`sys/module/hid/refcnt`	Cuenta de referencia del módulo hid (human interface device). Cuántos otros módulos usan a éste.
/sys/bus/usb/drivers	Carpeta con diferentes subcarpetas correspondientes a distintos drivers usados en el bus usb

Algunos ficheros se pueden usar para alterar el estado del hardware

Ejemplo: Controlar la velocidad máxima de los cores en un cierto ordenador. Observa las rutas donde aparecen los ficheros "scaling_max_freq". Escribiendo en ellos le estás diciendo al núcleo, la frecuencia máxima de las cpu (en este portátil había 2 cpu)

```
root@diezgb:/sys/devices/system/cpu/cpufreq/policy0# pwd
/sys/devices/system/cpu/cpufreq/policy0

root@diezgb:/sys/devices/system/cpu/cpufreq/policy0# ALTA=2100000
root@diezgb:/sys/devices/system/cpu/cpufreq/policy0# BAJA=1050000
# echo $ALTA > scaling_max_freq ; echo $ALTA >../policy1/scaling_max_freq
# echo $BAJA > scaling_max_freq ; echo $BAJA >../policy1/scaling_max_freq
```

¿Quién llena el directorio /sys?

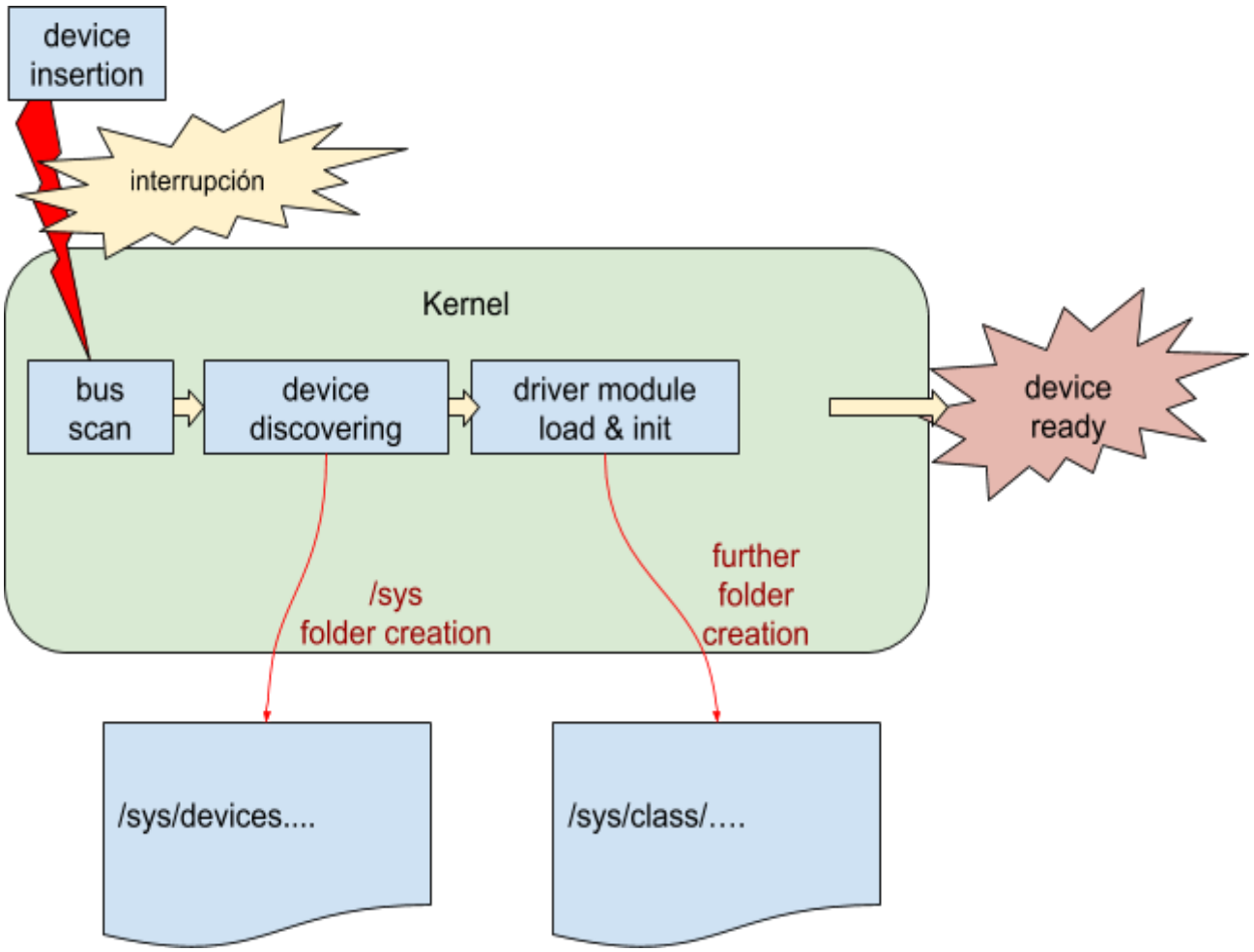
https://stackoverflow.com/questions/16849673/sysfs-entries-for-hotpluggable-devices ---> dudosa respuesta.

El directorio /sys es llenado por dos elementos:

- El driver de cada bus cuando detecta un dispositivo
- Los drivers de los dispositivos cuando son cargados e inicializados

Muy resumidamente:

1. El dispositivo es insertado en caliente (o durante el inicio se examina el hardware presente).
2. El driver del bus es capaz de extraer la información básica de identificación del hardware insertado/detectado (aunque éste siga inusable por falta de driver del dispositivo). Se crea entonces un directorio y algunos ficheros en /sys
3. **Algo** cargará en el núcleo el driver del dispositivo, y éste, en su primera ejecución, puede crear más carpetas y ficheros en /sys que no tienen la misma ubicación que los del punto 2. Ese "**algo**" será visto más adelante en este te



ma.

¿No está el directorio /dev hecho para algo similar?

El directorio /dev es muy antiguo en los sistemas unix y **no tiene la función de mostrar las características del hardware sino permitir usar los dispositivos a base de lecturas y escrituras directamente en ellos**. Cualquier periférico de entrada/salida es un fichero dentro de /dev, desde el ratón, hasta la pantalla, o los discos duros, una cámara, etc.

Por ejemplo, una manera de hacer una copia de seguridad de toda una memoria usb se consigue con

```
cat /dev/sdb | tar | gzip > copia.tgz
```

En este ejemplo, el comando cat lee de un fichero ("`/dev/sdb`") que **representa la información almacenada** en un disco duro. Esa información es tratada por los demás comandos para crear la copia de seguridad. Cada dispositivo referenciado desde el directorio /dev recibe el nombre de "nodo". En el comando anterior se lee todo un dispositivo usando uno nodo (o fichero)

El directorio `/dev` se mantiene y se usa de forma similar que hace años, pero la manera

en la que durante el inicio se crean elementos dentro de /dev es muy diferente hoy en día, y configurable.. Ahora /dev depende -entre otras cosas- de /sys y es creado y llenado de nodos a partir de /sys. La creación de nodos es responsabilidad ahora de un proceso (udev) y no del núcleo del sistema operativo. (iremos viendo todo esto).

Modalias en /sys

Las subcarpetas dentro de /sys contienen los modalias de cada dispositivo detectado. En cada carpeta que referencia a un dispositivo, hay un fichero (precisamente llamado "modalias") que contiene la línea descriptiva del hardware

```
cat $( find /sys -name modalias )
```

Seguramente, la información de cada modalias la descubre el driver del bus cuando consulta el dispositivo conectado a él, o el núcleo cuando interroga a la cpu por sus características, etc. Es decir, cada dispositivo, puede informar de su modalias al sistema operativo, aunque no exista driver de dicho dispositivo.

https://stackoverflow.com/questions/16849673/sysfs-entries-for-hotpluggable-devices

Udev

https://en.wikipedia.org/wiki/Udev

Udev es un programa que es responsable de varias cosas:

1. Cargar (u "ordenar cargar") los drivers del hardware detectado
2. Permitir acciones especiales cuando ocurre un cambio en algún hardware.
3. Nombrar los dispositivos de forma permanente y razonable. Así como poblar de nodos el directorio /dev

Udev ejecuta un servicio (proceso) llamado "udevd" en segundo plano que recibe notificaciones del núcleo del sistema operativo. Udev llama "eventos" a estas notificaciones. Estos eventos indican principalmente dos elementos:

- Cambio sufrido
- Modalias del hardware que ha sufrido el cambio.

Cuando udev recibe un evento reacciona cargando un driver (ya se ha explicado cómo descubrir qué driver), o aplicando "reglas udev" que desencadenan acciones. Es el administrador el que especifica estas reglas y acciones, que le permiten adaptar el funcionamiento del sistema a necesidades y requisitos básicos.

La carga de drivers es algo totalmente inherente a udev y no es necesario tratar aquí. Sin embargo, la configuración de reglas udev es donde quedan abiertas muchas opciones de

configuración y particularización de un sistema y requiere un tratamiento cuidadoso.

Una forma de observar este funcionamiento es monitorizar la actividad de udev. Existe un comando que muestra los eventos que el núcleo emite y udev recoge.

```
udevadm monitor
```

Ejemplo: Abre un terminal y teclea el comando udevadm monitor. inserta después un disco zusb y observa la salida

Observa que por cada evento aparecen dos líneas diferentes. Una corresponde al evento lanzado por el núcleo (KERNEL) y otra cuando el evento ha sido procesado por udev (UDEV)

Udev y /dev

https://en.wikipedia.org/wiki/Device_file#Unix_and_Unix-like_systems

Conforme Linux ha ido evolucionando, la forma de usar /dev ha ido planteado muchos problemas que han costado de solucionar porque es un directorio muy asentado y del que dependen muchas otras funciones. Udev se creó para solucionar varios problemas, algunos recientes causados por la evolución de linux, pero otros problemas que venían arrastrándose desde hace tiempo. El primer problema era el planteado por el directorio /dev

/dev estático (antiguo)

El principal y primer problema con /dev, es que el directorio se creaba y poblaba durante el arranque, con todas las entradas posibles (o "nodos", como también se les llama). Había una especie de lista pre-creada de nombres de fichero que debían aparecer en el directorio. Después de ser creado /dev, se asignaban los dispositivos físicos presentes a algunos de dichos nodos. Normalmente se creaban varios nodos que no llegaban a ser utilizados.

> Por ejemplo, dado que sólo podía haber 4 discos IDE, todos los computadores Linux tenían en este directorio cuatro nodos: "hda", "hdb", "hdc" y "hdd". Pero si un computador sólo tenía un disco duro y un cederrón sólo se usaban dos "hda" que era el disco principal y "hdb" que era el cederrón. "hdc" y "hdd" estaban ahí sin un dispositivo real asociado.

Había otros muchos nodos que no tenían un dispositivo físico real. Cada nuevo tipo de dispositivo popularizado era añadido a esa lista (por ejemplo una webcam), y aparecía en /dev la correspondiente entrada, aunque el computador no tuviese webcam.

Evolución de /dev

Este problema se solucionó y además, se dió a linux la posibilidad de crear nuevos nodos durante el funcionamiento normal de linux. Así, se facilitó la utilización de memorias usb y otros dispositivos contectables en caliente. El arreglo consistió en modificar el Núcleo y que éste detectase nuevos dispositivos conectados y crease consecuentemente entradas nuevas en /dev

Pero quedaba un problema sin fácil soluciòn. Veámoslo con un ejemplo:

Supón un computador con un solo disco SATA. Éste aparecerá en el nodo /dev/sda siempre. El usuario dispone de un lápiz usb o memoria usb que al ser insertada aparece en /dev/sdb. Para este usb, el usuario realiza siempre el montaje /dev/sdb1 (primera partición de la memoria usb) al directorio /media/miUSBFavorito y después trabaja con él normalmente. Supón que, con el tiempo, ha hecho un script para realizar el montaje y la copia de seguridad de los datos automáticamente a esta memoria usb. Estos son los eventos.

1. El usuario inserta el USB
2. El núcleo detecta el usb y crea /dev/sdb y /dev/sdb1
3. El usuario lanza el script hecho por él que realiza lo siguiente
    a. monta /dev/sdb1 en /media/miUSBFavorito
    b. copia algunos ficheros y carpetas de /home/usuario a /media/miUSBFavorito
4. El usuario desmonta el directorio y libera el usb:
    a. umount /dev/sdb1 o umount /media/miUSBFavorito

Cierto día llega una persona con otra memoria usb y la inserta antes de que este usuario inserte la suya. El núcleo no sabe que se trata de otra memoria usb diferente (no es preocupación del núcleo), por lo que decide asignarle /dev/sdb a dicha memoria. Cuando el usuario principal inserte su conocido USB, éste se creará en /dev/sdc (que es el siguiente libre) y llegará la confusión porque el script no está pensado para montar /dev/sdc en /media/miUSBFavorito.

Los deseos del usuario son que su USB aparezca siempre en /dev/sdb, pero el núcleo no desciende a atender los deseos de un mortal.

Antes de udev, el núcleo ignoraba los deseos de los usuarios y nombraba a los dispositivos con criterios simples pero rígidos. Los discos se iban descubriendo (o insertando) y se iban asignando consecutivamente a "hda", "hdb", "hdc", etc. (o "sda", "sdb"...).

Para dar solución a la nomeclatura de los dispositivos se decidió delegar en un

proceso de usuario ("udev") que el administrador pudiese manipular y configurar. El núcleo de Linux no debía complicarse hasta tal punto de decidir qué nombre poner a cada entrada de /dev de cada dispositivo. ¡Mejor que esto lo decida otro!

udev como proceso de usuario

"udev" viene de "Userspace DEV". Es algo así como "el antiguo directorio /dev pero gobernado por los deseos del usuario" (aquí "usuario" significa realmente "administrador")

Udev es principalmente un proceso (udevd) y por tanto no forma parte del núcleo. Esto es intencionado y buscado, y la idea que se persigue es:

> Delegar todo lo posible en procesos de usuario (con permisos de root) la gestión de los dispositivos, de su identificación, de las acciones a realizar cuando son insertados y de control de los privilegios de acceso

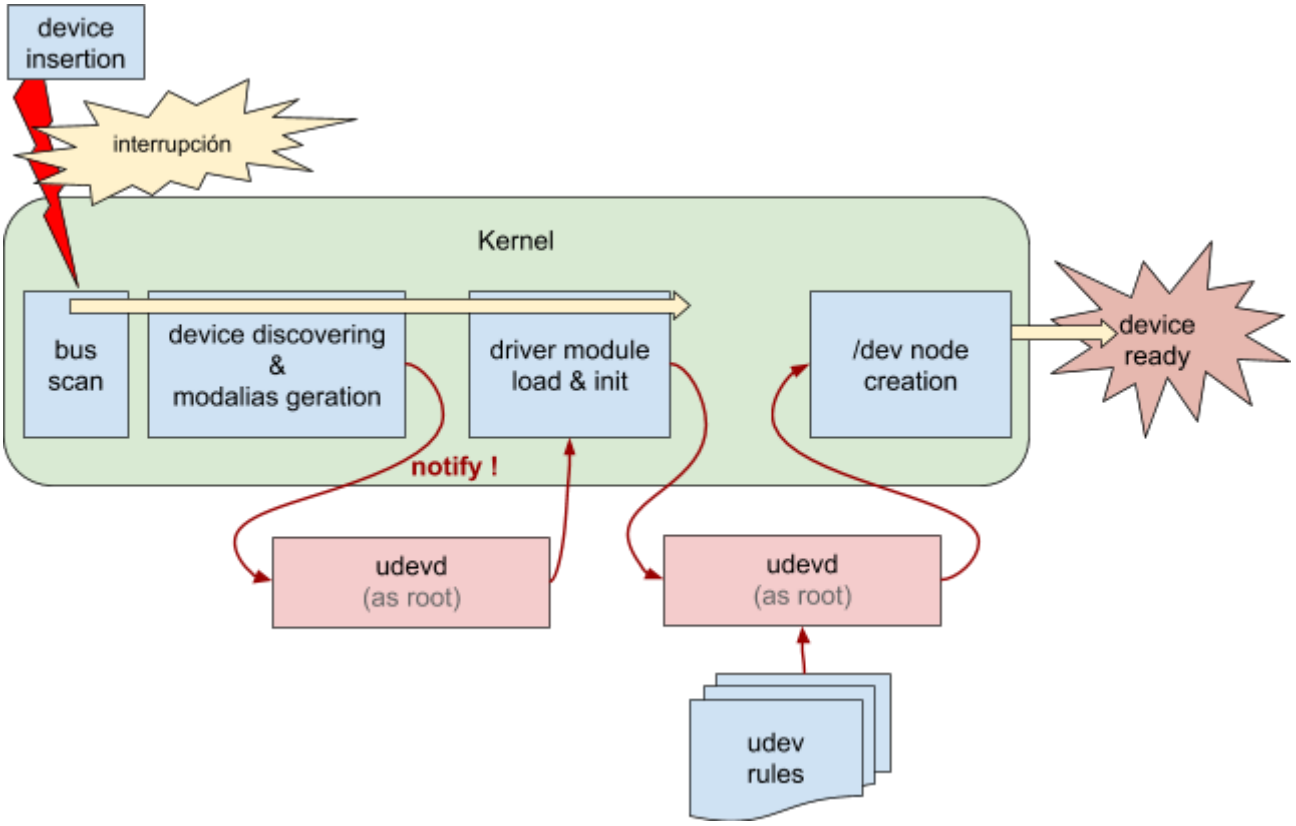
La intención es que el núcleo decida las menos cosas posibles, aunque evidentemente, sólo él será capaz de hacer ciertas cosas. Pero muchas de las acciones a realizar serán ordenadas por udev, no decididas por el núcleo

El núcleo avisa a udev y éste toma decisiones que transmite al núcleo

udev como dueño de /dev

Una de las tareas básicas de udev es ahora crear los nodos en el directorio /dev. Durante el arranque, conforme el núcleo va detectando dispositivos, va informando a udev, que -además de cargar los drivers- decide qué entradas o nodos deben crearse en /dev según unas reglas.

En una instalación de linux normal, se dejan creadas unas reglas que están bastante ajustadas a la necesidad del usuario. Por ejemplo se facilita enormemente la creación de puntos de montaje cuando se insertan memorias usb externas.



Reglas de udev

http://www.reactivated.net/writing_udev_rules.html

Como ya se ha introducido, udevd es un proceso que "escucha" al núcleo. Cada vez que ocurre un evento hardware o de drivers importante, el núcleo emite un aviso o notificación por un canal de comunicación llamado "netlink" ("netlink socket").

udev analizará la información que conlleva el aviso y consultará una lista de reglas para determinar qué se debe hacer con el dispositivo que ha provocado el evento. Las reglas vienen a determinar **qué se hace con cada dispositivo detectado**, y permiten diferenciar según tipos, función, modelo y fabricante, de los dispositivos. Así, un disco duro será nombrado acorde a un criterio, y una tarjeta de red seguirá otro distinto.

Las reglas, además, sirven para notificar el hardware al resto de procesos y sistemas. Esto es la base de lo que un usuario observa cuando inserta una memoria USB y aparece una ventana preguntándole qué desea hacer con el dispositivo, o se muestra la unidad en el explorador de archivos. Todo eso que ocurre, es consecuencia de alguna regla que se aplica para el USB insertado.

Esta respuesta ocurre gracias a varios elementos, udev el primero de ellos.

Las reglas deciden cómo se reacciona ante la presencia del hardware. Y dado que el usuario administrador puede crear, cambiar o eliminar reglas, se puede administrar el hardware ajustándolo a la necesidad del usuario. Vamos a descubrir las reglas

¿Dónde están escritas las reglas?

En una instalación habitual, existen reglas predefinidas por la distribución usada que suelen residir en `/lib/udev/rules.d`, o en `/usr/lib/udev/rules.d`. Pero si el administrador desea escribir sus propias reglas, la carpeta adicional destinada a ello es **`/etc/udev/rules.d`**. Los ficheros de reglas suelen tener la **extensión** "**`.rules`**".

En este caso es preferible crear allí (en /etc...) un nuevo fichero para cada regla a configurar. Y para ello, has de saber que los nombres de ficheros determinan el orden de procesamiento de los mismos, y por tanto, de las reglas a veces. Es por esto, que los nombres suelen empezar con un número que es un elemento fácil de usar para imponer un orden. Por ejemplo, podrías llamar `/etc/udev/rules.d/10-local.rules` al fichero que añades para poner tus reglas. Ya que 10 es un número pequeño (suelen nombrarse del 0 al 99) y te aseguras que se ejecutará el primero pero dejando margen para poner después otros ficheros que se debieran procesar antes que este (por ejemplo `05-miconf.rules`).

Las reglas en /etc/udev/rules.d se procesan antes que las /usr/lib/udev/rules.d y en caso de haber dos reglas con el mismo nombre en diferenes directorios, la que está en /etc tiene preferencia y se aplicará, dejando la regla con el mismo nombre de /usr.... o de /lib sin ser aplicada.

One device can be matched by more than one rule. This has it's practical advantages, for example, we can write two rules which match the same device, where each one provides its own alternate name for the device. Both alternate names will be created, even if the rules are in separate files. It is important to understand that udev will not stop processing when it finds a matching rule, it will continue searching and attempt to apply every rule that it knows about.

Especificaciòn de las reglas udev

En informática, la palabra "regla" se usa en muchos ámbitos. Y generalmente significa una indicación de las **acciones que deben llevarse a cabo cuando se cumplen ciertas condiciones**. Hay reglas en bases de datos, reglas de copias de seguridad, reglas en inteligencia artificial, reglas de acceso a un directorio, reglas que controlan la seguridad en un sistema. En general, en todas las reglas hay dos partes:

1. **condiciones...** que deben cumplirse
2. **acciones...** que se ejecutarán si se cumplen las condiciones

Sintaxis de las reglas (¿cómo se escriben?)

Las reglas de udev tienen una forma particularmente simple:

- Cada regla es una línea.
- Cada línea tiene varios pares clave-valor.
- Tanto las condiciones como las acciones son pares clave valor. Sintácticamente no se distinguen casi (es decir, sólo por cómo se escriben no las diferenciarás fácilmente)

¡Sí! Resulta raro que en la misma línea, sin mucha diferenciación haya condiciones y acciones, y que además, tengan una escritura muy similar. Vamos con un ejemplo

```
KERNEL=="hdb", NAME="mi_disco"
```

En esta regla hay dos pares clave-valor:

clave - valor	clave	valor	Tipo
`KERNEL=="hdb"`	`KERNEL`	`hdb`	**condición**
`NAME="mi_disco"`	`NAME`	`mi_disco`	**acción**

Observa que en la condición, se ha usado el operador "==" que es un operador de comparación habitualmente, se usa para comprobar una igualdad. Mientras que en la acción, se usa el operador de asignación "=" para dar valor a algo. **Esto es lo que determina qué es una condición y qué es una acción, dentro de la regla.**

En terminología udev, el par **clave-valor** es llamado "**key-value**" y se usa en las condiciones y en las acciones.

- Las **condiciones** también se les llama "**match**" key-value. "match" en inglés es *coincidir*

y tiene el sentido de que el valor del dispositivo coincida con el de la regla para dispararla

- Las **acciones** reciben el nombre de **assignment** key-value , ya que se asigna un valor o parámetro al elemento que representa al dispositivo.

```
KERNEL=="sd*",VENDOR=="TOSHIBA",SERIE=="125421254",NAME="discocopia"
```

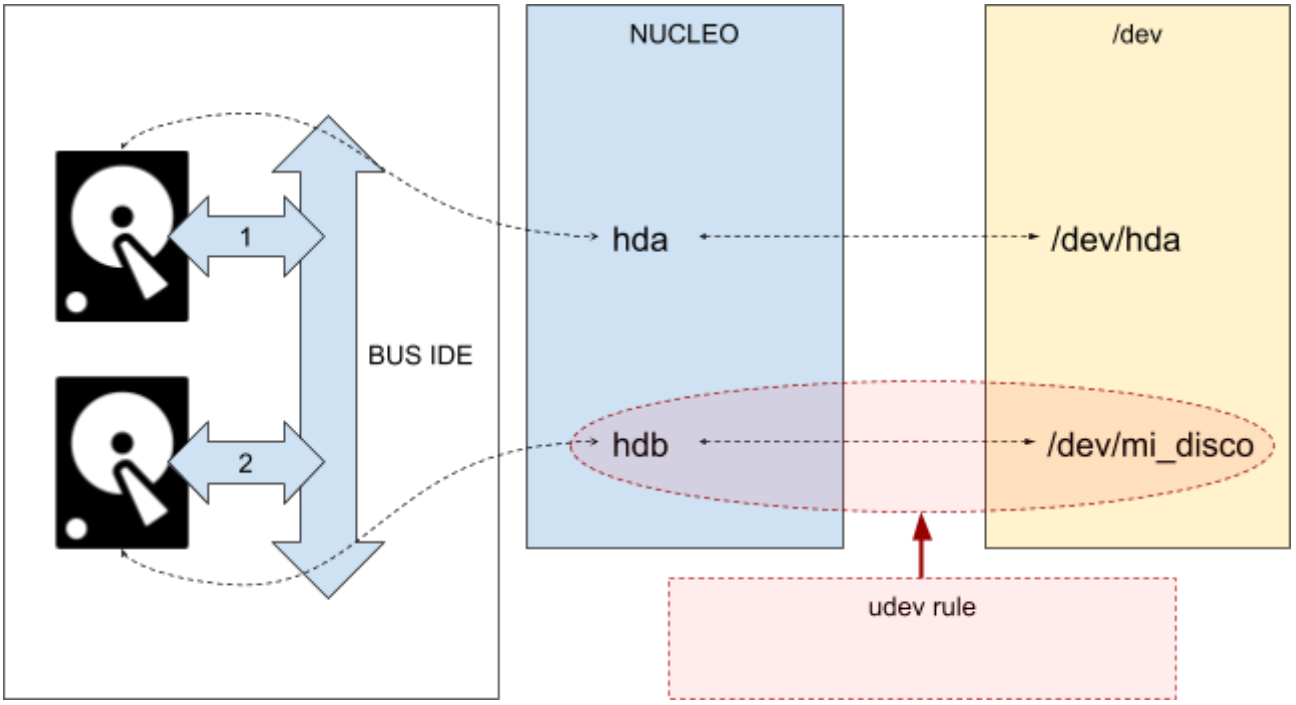
Semántica de las reglas (¿Qué significan?)

Para entender qué hace una regla, hay que entender qué significa o controla cada clave. En el caso de la regla anterior

```
KERNEL=="hdb", NAME="mi_disco"
```

- `KERNEL` hace referencia al nombre del dispositivo para el núcleo (que coincide con el habitual tradicional visible en /dev, pero aquí es el nombre para el núcleo, antes de que aparezca en /dev)
- `NAME` hace referencia al nombre del nodo que aparecerá en /dev

Por tanto estamos diciendo que para el disco hdb, se creará un nodo en /dev llamado mi_disco. Es decir `/dev/mi_disco` referenciará al segundo disco IDE



Otro ejemplo

```
KERNEL=="hdb", DRIVER=="ide-disk", SYMLINK+="sparedisk"
```

En este caso, se identifica a un dispositivo que para el núcleo se llama "hdb", pero además,

esta regla exige como condición adicional que el driver de dicho dispositivo sea "ide-disk" (dejaríamos fuera otros discos, sata por ejemplo).

La parte de las acciones no indica "NAME" como antes (por lo que el nombre por defecto de "hdb" se va a usar igualmente para designar al nodo de este dispositivo "`/dev/hdb`"). Con SYMLINK+ se está asignando otro nodo adicional llamado "sparedisk" (`/dev/sparedisk` y `/dev/hdb` serán el mismo dispositivo)

> Para que una regla tenga sentido, debe tener una **match key-value** y una **assignment key-value**.

Match key genéricas vs atributos

En la parte de las condiciones se pueden especificar atributos que identifiquen a un hardware en particular, para ello están las match key pensadas. La pregunta ahora es ¿Con qué claves formar condiciones para un hardware deseado? ¿Qué clave y valor puedo usar para identificar éste hardware concreto?

Las match keys se pueden agrupar en dos:

1. **Genéricas**, que todo dispositivo tiene: nombre, sistema, driver
2. **Específicas** de cada tipo de dispositivo, como modelo, número de serie, versión del hardware, etc.

El descubrimiento o inserción de un hardware comporta eventos y acciones en el núcleo:

- Se nombra al dispositivo según varios factores
- El dispositivo se añade a los dispositivos de un subsistema
- Se carga un driver
- Se reconocen una serie de atributos y características específicas del dispositivo

Cada uno de estos cuatro elementos tiene su palabra clave

`KERNEL`	Se compara el nombre dado en la regla con el que el núcleo le pone al dispositivo
`SUBSYSTEMS`	Se filtra por el subsistema (red, discos duros, etc)
`DRIVER`	Se coteja el driver (no se decide qué driver cargar)
`ATTRS`	Se coteja el valor de algún atributo del hardware introducido

Un ejemplo de regla (simple) que contiene una condición basada en un atributo del dispositivo podría ser:

Selección de match key

Los atributos del hardware con los que podemos especificar condiciones en las reglas son diferentes en cada dispositivo. Es posible que en algún momento llegues a pensar "...Y para este dispositivo, *¿¡ Qué atributos puedo usar en una regla?!*". Existen distintas formas de averiguar las claves que un dispositivo expone, desde usar el comando udev en sus diferentes variantes, hasta examinar el directorio /sys del dispositivo.

El primer comando ideal es el ofrecido por el propio udev: `udevadm` que tiene varios subcomandos, de los cuales "`info`" es el necesario.

El comando más directo para obtener información sobre los atributos usables en una regla es (por ejemplo, para el dispositivo `/dev/input/js0`) (dos variantes iguales:

```
udevadm info --attribute-walk /dev/input/js0
udevadm info -a -p /dev/input/js0
```

El anterior comando asume que conoces la ruta del dispositivo que cuelga de /dev. Pero realmente el comando udevadm info obtiene la información del directorio /sys. Es posible averiguar qué directorio `/sys` corresponde a un dispositivo del que conocemos su entrada en /dev. Toma el siguiente ejemplo para ello

```
udevadm info -q path -n /dev/sdc
```

La salida del anterior comando es algo así:

```
devices/pci0000:00/0000:00:0d.0/ata3/host2/target2:0:0/2:0:0:0/block/sdc
```

Ahora, con esa línea se puede invocar al comando udevadm info para obtener el listado de todas las claves, del dispositivo y de todos sus dispositivos padres (en la jerarquía de conexionado)

```
udevadm info -a -p devices/pci00...:0/block/sdc
```

La salida de ambos comandos es similar a la siguiente captura (para el caso de un ratón):

```
looking at device '/devices/platform/i8042/serio1/input/input17/mouse1':
      KERNEL=="mouse1"
      SUBSYSTEM=="input"
      DRIVER==""

  looking at parent device '/devices/platform/i8042/serio1/input/input17':
      KERNELS=="input17"
      SUBSYSTEMS=="input"
      DRIVERS==""
```

```
      ATTRS{uniq}==""
      ATTRS{properties}=="1"
      ATTRS{phys}=="isa0060/serio1/input0"
      ATTRS{name}=="SynPS/2 Synaptics TouchPad"

  looking at parent device '/devices/platform/i8042/serio1':
      KERNELS=="serio1"
      SUBSYSTEMS=="serio"
      DRIVERS=="psmouse"
      ATTRS{resetafter}=="5"
      ATTRS{resolution}=="200"
      ATTRS{description}=="i8042 AUX port"
      ATTRS{firmware_id}=="PNP: SYN0150 SYN0100 SYN0002 PNP0f13"
      ATTRS{protocol}=="SynPS/2"
      ATTRS{rate}=="80"
      ATTRS{bind_mode}=="auto"
      ATTRS{resync_time}=="0"

  looking at parent device '/devices/platform/i8042':
      KERNELS=="i8042"
      SUBSYSTEMS=="platform"
      DRIVERS=="i8042"
      ATTRS{driver_override}=="(null)"

  looking at parent device '/devices/platform':
      KERNELS=="platform"
      SUBSYSTEMS==""
      DRIVERS==""
```

Lo que ves es un listado de todos los atributos o pares clave-valor que identifican al dispositivo y a los dispositivos padre del preguntado.

Pero para usar el comando anterior debes saber la ruta en /dev o en /sys. **¿Qué pasa si no puedes averiguar fácilmente la ruta en /dev o /sys?** En este caso es imposible usar el comando udevadm anterior. Lo que haremos será utilizar la monitorización de eventos para averiguar dicha ruta o para directamente descubrir los atributos a usar. La idea es la siguiente

1. Empezar a monitorizar los eventos hw con el comando udevadm monitor
2. insertar o activar el dispositivo
3. Observar la salida de la monitorización de eventos y elegir la información necesaria:
    a. La ruta del dispositivo dentro de /sys
    b. Algún atributo si se usa el parámetro `--env` en el comando `udevadm monitor`

La opción --env es muy poderosa y muestra muchísima información

Finalmente, si el dispositivo no se puede insertar o extraer, se utilizará el comando lspci para obtener la información que nos llevará a la ruta dentro de la carpeta /sys, y con ella, el comando udevadm nos mostrará los atributos usables. Esto se detallará más adelante en el

comando lspci

https://unix.stackexchange.com/questions/60078/find-out-which-modules-are-associated-with-a-usb-device

¿Qué acciones puedo desencadenar con una regla?

http://www.reactivated.net/writing_udev_rules.html

Veremos a continuación qué decisiones y cambios puede establecer una regla.

Nombrar dispositivos

La función básica y primitiva de las reglas es establecer unos nombres concretos a los dispositivos. Esto se controla con la clave "NAME=*nombre*". Así la siguiente regla creará el dispositivo /dev/mi_disco a un dispositivo que para el núcleo tendría la identificación de /dev/sdb.

```
KERNEL=="sdb", NAME="mi_disco"
```

Otro uso posible, por ejemplo, es renombrar la tarjeta de red, que recientemente viene nombrada como `enp0S3` en la mayoría de sistemas a `eth0` como era conocida antes. (esta regla no está probada y no sé si funciona, se pone como ejemplo instructivo)

```
KERNEL=="enp0s3", NAME="eth0"
```

Establecer permisos sobre dispositivos

Udev permite establecer permisos sobre qué usuario o grupo puede utilizar cierto dispositivo

La clave GROUP establece qué grupo Unix será el dueño del nodo /dev que se crea para el dispositivo. En el siguiente ejemplo, todos los dispositivos de tipo framebuffer (¿webcam?) son entregados al grupo "video",

```
KERNEL=="fb[0-9]*", NAME="fb/%n", SYMLINK+="%k", GROUP="video"
```

La clave OWNER (habitualmente menos útil) establece qué usuario será el dueño del dispositivo. Imagina que quieres que juanito sea el dueño de las disqueteras.

```
KERNEL=="fd[0-9]*", OWNER="juanito"
```

Una cosa es el dueño y el grupo de un nodo, y otra los permisos que sobre él se establecen. ¿De qué sirve asignar un cierto grupo si después los permisos son 777 o 666 y todos pueden acceder? Udev crea los nodos con permisos 660, es decir, sólo dueño y grupo pueden leer y escribir únicamente. Si necesitas otro tipo de permisos, utiliza la clave MODE para indicar los permisos con los que se crea un nodo. Por ejemplo, el siguiente dispositivo tiene permisos de

lectura y escritura para todo el mundo. :

```
KERNEL=="inotify", NAME="misc/%k", SYMLINK+="%k", MODE="0666"
```

Nombrar dispositivos con programas externos

Under some circumstances, you may require more flexibility than standard udev rules can provide. In this case, you can ask udev to run a program and use the standard output from that program to provide device naming.

To use this functionality, you simply specify the absolute path of the program to run (and any parameters) in the PROGRAM assignment, and you then use some variant of the %c substitution in the NAME/SYMLINK assignments.

The following examples refer to a fictional program found at /bin/device_namer. device_namer takes one command line argument which is the kernel name for the device. Based upon this kernel name, device_namer does its magic and produces some output to the usual stdout pipe, split into several parts. Each part is just a single word, and parts are separated by a single space.

In our first example, we assume that device_namer outputs a number of parts, each one to form a symbolic link (alternative name) for the device in question.

```
KERNEL=="hda", PROGRAM="/bin/device_namer %k", SYMLINK+="%c"
```

The next example assumes that device_namer outputs two parts, the first being the device name, and the second being the name for an additional symbolic link. We now introduce the %c{N} substitution, which refers to part N of the output:

```
KERNEL=="hda",
PROGRAM="/bin/device_namer %k", NAME="%c{1}", SYMLINK+="%c{2}"
```

The next example assumes that device_namer outputs one part for the device name, followed by any number of parts which will form additional symbolic links. We now introduce the %c{N+} substitution, which evaluates to part N, N+1, N+2, ... until the end of the output.

```
KERNEL=="hda", PROGRAM="/bin/device_namer %k", NAME="%c{1}",
SYMLINK+="%c{2+}"
```

Output parts can be used in any assignment key, not only NAME and SYMLINK. The example below uses a fictional program to determine the Unix group which should own the device:

```
KERNEL=="hda", PROGRAM="/bin/who_owns_device %k", GROUP="%c"
```

Ejecutar programas asociados a eventos

Yet another reason for writing udev rules is to run a particular program when a device is connected or disconnected. For example, you might want to execute a script to automatically

download all of your photos from your digital camera when it is connected.

Do not confuse this with the PROGRAM functionality described above. PROGRAM is used for running programs which produce device names (and they shouldn't do anything other than that). When those programs are being executed, the device node has not yet been created, so acting upon the device in any way is not possible.

The functionality introduced here allows you to run a program after the device node is put in place. This program can act on the device, however it must not run for any extended period of time, because udev is effectively paused while these programs are running. One workaround for this limitation is to make sure your program immediately detaches itself.

Here is an example rule which demonstrates the use of the RUN list assignment:

```
KERNEL=="sdb", RUN+="/usr/bin/my_program"
```

When `/usr/bin/my_program` is executed, various parts of the udev environment are available as environment variables, including key values such as SUBSYSTEM. You can also use the ACTION environment variable to detect whether the device is being connected or disconnected - ACTION will be either "add" or "remove" respectively.

udev does not run these programs on any active terminal, and it does not execute them under the context of a shell. Be sure to ensure your program is marked executable, if it is a shell script ensure it starts with an appropriate shebang (e.g. #!/bin/sh), and do not expect any standard output to appear on your terminal.

Environment interaction

udev provides an ENV key for environment variables which can be used for both matching and assignment.

In the assignment case, you can set environment variables which you can then match against later. You can also set environment variables which can be used by any external programs invoked using the techniques mentioned above. A fictional example rule which sets an environment variable is shown below.

```
KERNEL=="fd0", SYMLINK+="floppy", ENV{some_var}="value"
```

In the matching case, you can ensure that rules only run depending on the value of an environment variable. Note that the environment that udev sees will not be the same user environment as you get on the console. A fictional rule involving an environment match is shown below.

```
KERNEL=="fd0", ENV{an_env_var}=="yes", SYMLINK+="floppy"
```

The above rule only creates the `/dev/floppy` link if `$an_env_var` is set to "yes" in udev's environment.

Integración de udev y systemd

Nota:Existe un ejercicio explicado que despliega todos los conceptos aquí explicados, es interesante y "chulo"

Existe una acción particularmente poderosa de una regla udev que consiste en activar un servicio vía systemd.

La acción RUN de una regla udev vista antes está pensada para lanzar un ejecutable asociado a un evento de inserción de dispositivo. Sin embargo, este ejecutable debe terminar pronto. Udev que es el demonio que lo lanza, lo monitoriza y le permite un tiempo breve de vida (de otra forma, dejaríamos detenida la máquina)

Si deseamos lanzar un proceso que tarde más tiempo, o por ejemplo, activar un cierto servicio a la introducción de un comando, entonces deberemos asociar la regla a un servicio systemd basado en una plantilla (template). El funcionamiento está basado en la clave `ENV{SYSTEMD_WANTS}` y su funcionamiento es como sigue.

1. Una regla udev se dispara y entre las acciones el administrador ha incluido un elemento similar al siguiente

   ```
   ENV{SYSTEMD_WANTS}="miservicio@%k.service"
   ```

2. (quizá falso pero no dañino) Esta acción intenta activar una unit de tipo servicio de systemd llamada "miservicio.service". (El "%k" es una variable de udev que será sustituida, pero ahora mismo, óbviala)

3. Systemd no encuentra esta unit concreta, pero sí otra que se llama "miservicio@.service". El "@" en el nombre indica que se trata de un template o plantilla. Esto se utiliza para crear sobre la marcha nuevas units.

4. Systemd crea una nueva unit "transient" (tranistoria) y le pone un nombre en el que se inserta entre el símbolo "@" y el "." de ".service" lo que aparece en la regla de udev, que es en sí una variable. Esta variable es el nombre del dispositivo para el núcleo.

   Por ejemplo, si se ha insertado un usb y ha recibido el nombre de sdc, entonces la unit recibirá el nombre de miservicio@sdc.service y esta unit será creada a partir del fichero /lib/systemd/system/miservicio@.service

5. Dentro de la plantilla, las variables "%i" y "%I" hacen referencia a la parte variable del nombre de la unit. En el caso anterior, el nombre del dispositivo para el núcleo.

Ahora debes crear el servicio (o plantilla) en el directorio de servicios systemd

<revisar ubicación de este fichero, en el ejercicio no está ahí, esto es sólo una simplificación>

```
cat >/etc/systemd/system/my-service@.service <<'EOF'
[Unit]
```

```
Description=My Service
[Service]
Type=simple
ExecStart=/path/to/your/script %I
EOF
```

And add `ENV{SYSTEMD_WANTS}="my-service@%k.service"` to the udev rule. Donde %k es el nombre del dispositivo para el núcleo, para el cual se está disparando esta regla.

Así, el atributo indicado se concretará con un nombre por ejemplo:

```
my-service@sda.service
```

Que dará lugar a que en la unit, el parámetro `-I` se sustituya por `"sda"`

This will run `/path/to/your/script` and pass it the path to the device that has just appeared.

Probar y Relanzar las reglas

Si has hecho una regla y la quieres probar deberán, en principio, ocurrir dos cosas,

1. La regla se debe cargar
2. El dispositivo debe ser detectado. Si ya está insertado y funcionando, no se detecta

El primer paso se realiza automáticamente porque udev, detecta cambios en el directorio de las reglas y reacciona inmediatamente recargándolas. Aún así, cabe la posibilidad de recargar las reglas mediante el comando

```
udevadm control --reload-rules
```

Algunas reglas reaccionan a cambios en el hardware y si cambias la regla, habrá que extraer y reinsertar el dispositivo. si esto es imposible, el siguiente comando reaplica las reglas a dispositivos existentes como si éstos hubiesen sido insertados

```
udevadm trigger
```

udevadm test

Si sabes la ruta "top-level" del dispositivo en `/sys`, puedes usar `udevadm test` para mostrar las acciones que udev tomaría con dicho dispositivo al ser insertado o detectado. Esto te puede ayudar a depurar tus reglas. Por ejemplo, supón que quieres ver el efecto de las reglas que se disparan al introducir o detectar una tarjeta de sonido.

```
# udevtest /class/sound/dsp
main: looking at device '/class/sound/dsp' from subsystem 'sound'
```

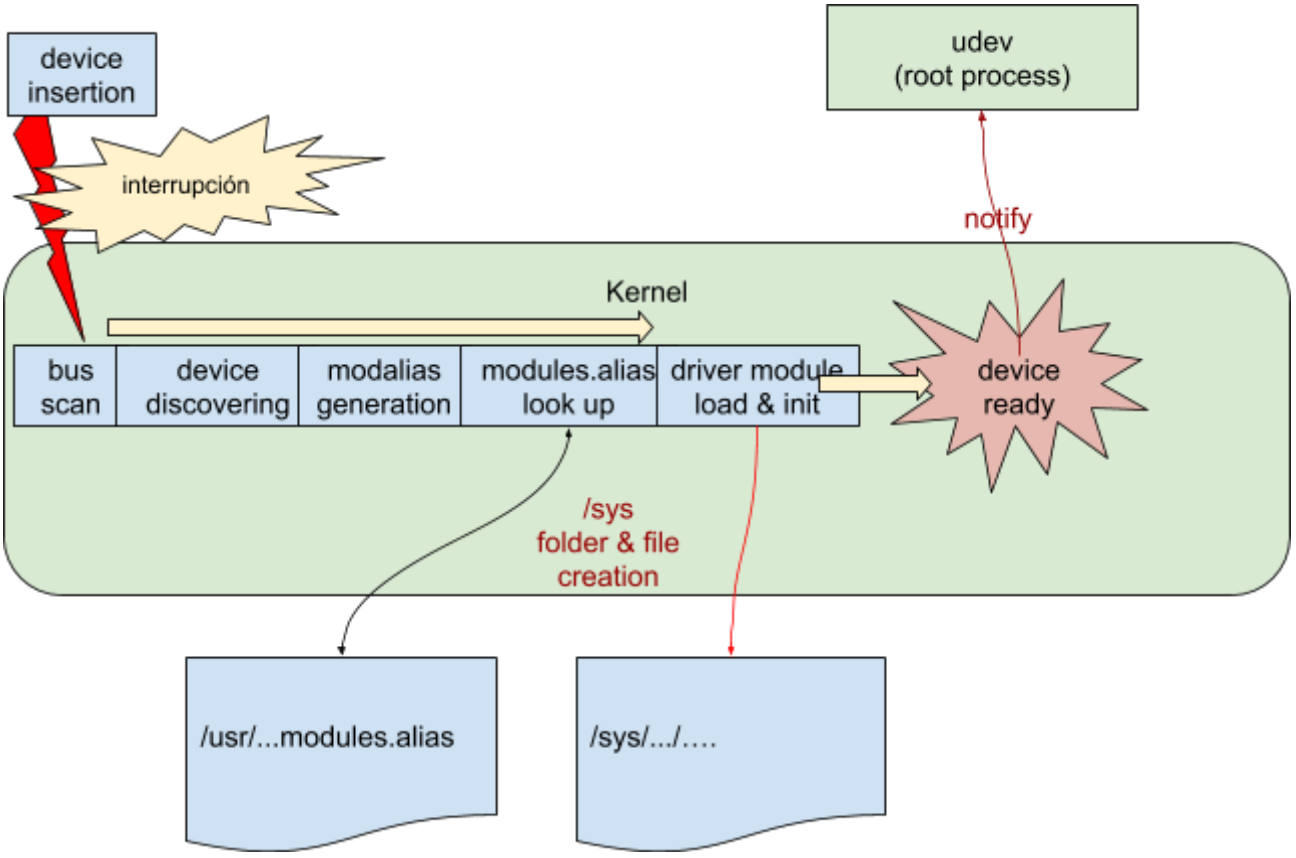
```
udev_rules_get_name: add symlink 'dsp'
udev_rules_get_name: rule applied, 'dsp' becomes 'sound/dsp'
udev_device_event: device '/class/sound/dsp' already known, remove possible
symlinks
udev_node_add: creating device node '/dev/sound/dsp', major = '14', minor =
'3', mode = '0660', uid = '0', gid = '18'
udev_node_add: creating symlink '/dev/dsp' to 'sound/dsp'
```

Observa que el prefijo /sys ha sido eliminado de la línea de comandos del comando. Esto ocurre porque `udevadm test` trabaja en rutas de dispositivos. Además, `udevadm test` es una herramienta de depuración o comprobación pura, no realiza ninguna acción real en los nodos de los dispositivos a pesar de lo que la salida de su ejecución muestra

https://sites.google.com/site/itmyshare/system-admin-tips-and-tools/udevadm---useage-examples

https://superuser.com/questions/881929/udev-pci-device-rule-is-not-working

https://forums.opensuse.org/showthread.php/485261-Script-run-from-udev-rule-gets-killed-shortly-after-start



Descubrir y listar el hardware

lspci

lspci (LIst PCI) lista los dispositivos conectados en el bus pci. El listado puede ser algo simple mostrando un dispositivo por línea:

```
[arch-fijo ~]# lspci
00:00.0 Host bridge: Advanced Micro Devices, Inc. [AMD/ATI] RD9x0/RX980 Host Bridge (rev 02)
00:02.0 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] RD890/RD9x0/RX980 PCI to PCI bridge (PCI Express GF
00:04.0 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] RD890/RD9x0/RX980 PCI to PCI bridge (PCI Express GP
00:09.0 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] RD890/RD9x0/RX980 PCI to PCI bridge (PCI Express GP
00:11.0 SATA controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 SATA Controller [AHCI mode]
00:12.0 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB OHCI0 Controller
00:12.2 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB EHCI Controller
00:13.0 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB OHCI0 Controller
00:13.2 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB EHCI Controller
00:14.0 SMBus: Advanced Micro Devices, Inc. [AMD/ATI] SBx00 SMBus Controller (rev 42)
00:14.1 IDE interface: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 IDE Controller (rev 40)
00:14.3 ISA bridge: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 LPC host controller (rev 40)
00:14.4 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] SBx00 PCI to PCI Bridge (rev 40)
00:14.5 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB OHCI2 Controller
00:15.0 PCI bridge: Advanced Micro Devices, Inc. [AMD/ATI] SB700/SB800/SB900 PCI to PCI bridge (PCIE port 0)
00:16.0 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB OHCI0 Controller
00:16.2 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB EHCI Controller
00:18.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Family 10h Processor HyperTransport Configuration
00:18.1 Host bridge: Advanced Micro Devices, Inc. [AMD] Family 10h Processor Address Map
00:18.2 Host bridge: Advanced Micro Devices, Inc. [AMD] Family 10h Processor DRAM Controller
00:18.3 Host bridge: Advanced Micro Devices, Inc. [AMD] Family 10h Processor Miscellaneous Control
00:18.4 Host bridge: Advanced Micro Devices, Inc. [AMD] Family 10h Processor Link Control
01:00.0 VGA compatible controller: Advanced Micro Devices, Inc. [AMD/ATI] Cape Verde XT [Radeon HD 7770/8760 /
01:00.1 Audio device: Advanced Micro Devices, Inc. [AMD/ATI] Cape Verde/Pitcairn HDMI Audio [Radeon HD 7700/78
02:00.0 USB controller: Etron Technology, Inc. EJ168 USB 3.0 Host Controller (rev 01)
03:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Co
 06)
```

Puedes querer precisar la información de dos maneras:

- Más detalle (opción -v, -vv, -vvv)
- Sólo de algún dispositivo. (opción -s ID_DISPOSITIVO)

La siguiente captura muestra la salida del comando ls -vvv para un dispositivo de tipo controlador de USB. (pruébalo en tu computador)

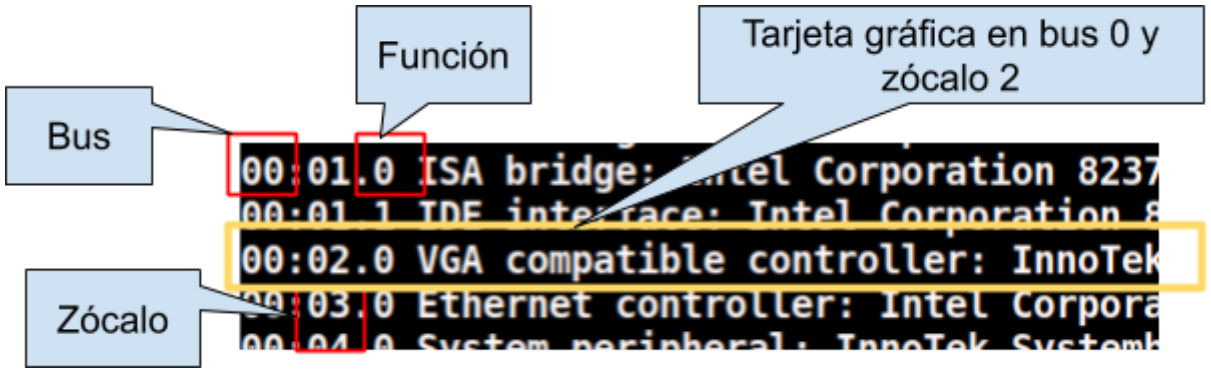
```
00:16.2 USB controller: Advanced Micro Devices, Inc. [AMD/ATI] SB7x0/SB8x0/SB9x0 USB EHCI Cont
        Subsystem: Gigabyte Technology Co., Ltd Device 5004
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV+ VGASnoop- ParErr- Stepping- SERR- Fa
        Status: Cap+ 66MHz+ UDF- FastB2B+ ParErr- DEVSEL=medium >TAbort- <TAbort- <MAbort- >SE
        Latency: 32, Cache Line Size: 64 bytes
        Interrupt: pin B routed to IRQ 17
        NUMA node: 0
        Region 0: Memory at fdff8000 (32-bit, non-prefetchable) [size=256]
        Capabilities: [c0] Power Management version 2
                Flags: PMEClk- DSI- D1+ D2+ AuxCurrent=0mA PME(D0+,D1+,D2+,D3hot+,D3cold-)
                Status: D0 NoSoftRst- PME-Enable- DSel=0 DScale=0 PME-
                Bridge: PM- B3+
        Capabilities: [e4] Debug port: BAR=1 offset=00e0
        Kernel driver in use: ehci-pci
        Kernel modules: ehci_pci
```

Los dispositivos pci están categorizados y jerarquizados en :

- dominio (generalmente no se usa)
- bus

- zócalo (slot)
- función.

De forma que por ejemplo, una tarjeta gráfica puede estar ubicada en el bus 00, en el zócalo 02 y constituye la función 0. En este ejemplo, el comando devolvería el siguiente resultado:



Para identificar en un comando a un dispositivo y obtener más información usamos esta información con el parámetro -s

```
lspci -s 00:02.0 -vvv
```

```
nacho@servidorLDAP:~$ lspci -s 00:02.0 -vvv
00:02.0 VGA compatible controller: InnoTek Systemberatung GmbH VirtualBox
Graphics Adapter (prog-if 00 [VGA controller])
    Control: I/O+ Mem+ BusMaster- SpecCycle- MemWINV- VGASnoop- ParErr-
Stepping- SERR- FastB2B- DisINTx-
    Status: Cap- 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort-
<MAbort- >SERR- <PERR- INTx-
    Interrupt: pin A routed to IRQ 18
    Region 0: Memory at e0000000 (32-bit, prefetchable) [size=32M]
    [virtual] Expansion ROM at 000c0000 [disabled] [size=128K]
    Kernel driver in use: vboxvideo
    Kernel modules: vboxvideo
```

Esta identificación de dispositivo también nos lleva directamente al directorio /sys donde se describe el dispositivo.

```
/sys/devices/pci0000:00/0000:00:02.0
```

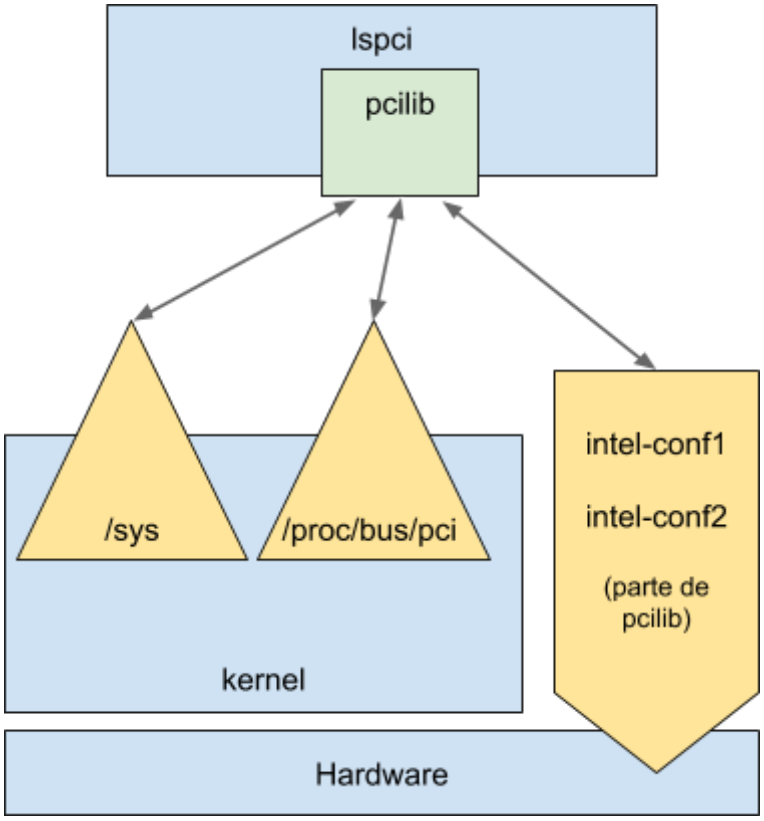
udevadm info `/sys/devices/pci0000:00/0000:00:02.0` nos dará todos los parámetros usables en las condiciones de las reglas

Método de acceso

El parámetro "-A" controla el método de acceso a la información sobre el hardware (es decir: "¿Cómo averigua el comando la información que se muestra al usuario?"). En principio, se

accede al sistema sysfs (el directorio /sys). Bajo este modo, el comando es una ventana al directorio /sys e interpreta su contenido para mostrarlo al usuario cómodamente.

Pero el comando puede acceder a la información del hardware mediante otros métodos que no necesitan /sys . Concretamente, intel proporciona librerías de acceso e inspección del hardware (pcilib), y ademaś, se puede usar el directorio /proc/bus/pci,



- `lspci -A linux-sysfs  # por defecto`
- `lspci -A linux-proc`
- `lspci -A intel-conf1`
- `lspci -A intel-conf2`

Averiguar qué driver maneja qué dispositivo.

En muchas ocasiones es interesante saber qué driver (o módulo) es el que se está usando para gestionar algún dispositivo. Quizá lo interesante sea simplemente ver si un dispositivo tiene algún módulo y descartar que no hay driver instalado para él. Para estos casos se usa el parámetro "-k"

```
lspci -k
```

lsusb

lsusb es un comando muy similar a lspci, pero centrado en el bus **usb** únicamente. Ten

en cuenta que un usuario suele cambiar dispositivos usb cada dos por tres, mientras que los dispositivos pci no se tocan, a veces nunca en la vida de un ordenador. Es el bus usb en el que se suelen conectar y desconectar dispositivos con frecuencia. Este comando puede ser más útil.

La invocación simple del comando muestra una lista de dispositivos usb

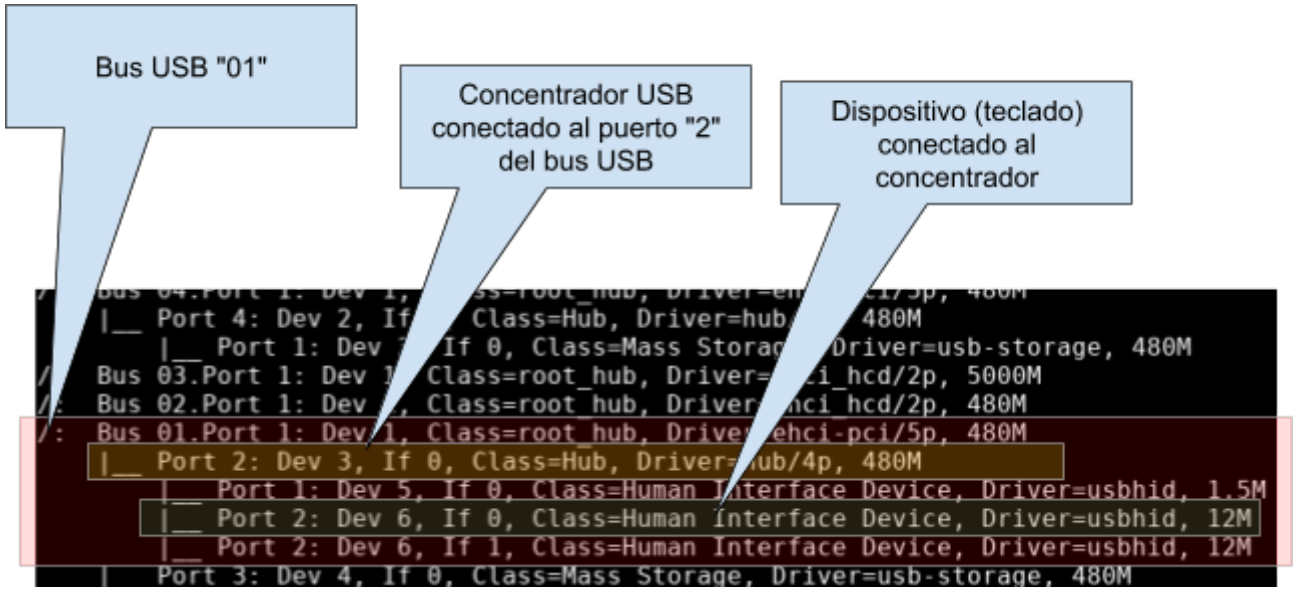
```
[arch-fijo ~]# lsusb
Bus 005 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 009 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 008 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 004 Device 003: ID 05e3:0749 Genesys Logic, Inc.
Bus 004 Device 002: ID 05e3:0610 Genesys Logic, Inc. 4-port hub
Bus 004 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 007 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
Bus 001 Device 004: ID 048d:1336 Integrated Technology Express, Inc. SD/MM
C Cardreader
Bus 001 Device 006: ID 195d:2033 Itron Technology iONE
Bus 001 Device 005: ID 1bcf:0002 Sunplus Innovation Technology Inc.
```

Jerarquía de buses usb

El bus usb es ampliable con "hubs". Estos concentradores usb son externos o van incorporados en algún dispositivo (por ejemplo, un teclado con conectores usb para comodidad del usuario). La salida habitual del comando no muestra la jerarquía de buses y dispositivos. Usando el parámetro **-t** se puede conseguir una mejor visualización del árbol de dispositivos.

```
[arch-fijo ~]# lsusb -t
/:  Bus 09.Port 1: Dev 1, Class=root_hub, Driver=ohci-pci/4p, 12M
/:  Bus 08.Port 1: Dev 1, Class=root_hub, Driver=ohci-pci/2p, 12M
/:  Bus 07.Port 1: Dev 1, Class=root_hub, Driver=ohci-pci/5p, 12M
/:  Bus 06.Port 1: Dev 1, Class=root_hub, Driver=ohci-pci/5p, 12M
    |__ Port 1: Dev 12, If 0, Class=Audio, Driver=snd-usb-audio, 12M
    |__ Port 1: Dev 12, If 1, Class=Audio, Driver=snd-usb-audio, 12M
    |__ Port 1: Dev 12, If 2, Class=Audio, Driver=snd-usb-audio, 12M
/:  Bus 05.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/4p, 480M
/:  Bus 04.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/5p, 480M
    |__ Port 4: Dev 2, If 0, Class=Hub, Driver=hub/4p, 480M
        |__ Port 1: Dev 3, If 0, Class=Mass Storage, Driver=usb-storage, 480M
/:  Bus 03.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/2p, 5000M
/:  Bus 02.Port 1: Dev 1, Class=root_hub, Driver=xhci_hcd/2p, 480M
/:  Bus 01.Port 1: Dev 1, Class=root_hub, Driver=ehci-pci/5p, 480M
    |__ Port 2: Dev 3, If 0, Class=Hub, Driver=hub/4p, 480M
        |__ Port 1: Dev 5, If 0, Class=Human Interface Device, Driver=usbhid, 1.5M
        |__ Port 2: Dev 6, If 0, Class=Human Interface Device, Driver=usbhid, 12M
        |__ Port 2: Dev 6, If 1, Class=Human Interface Device, Driver=usbhid, 12M
    |   Port 3: Dev 4, If 0, Class=Mass Storage, Driver=usb-storage, 480M
```

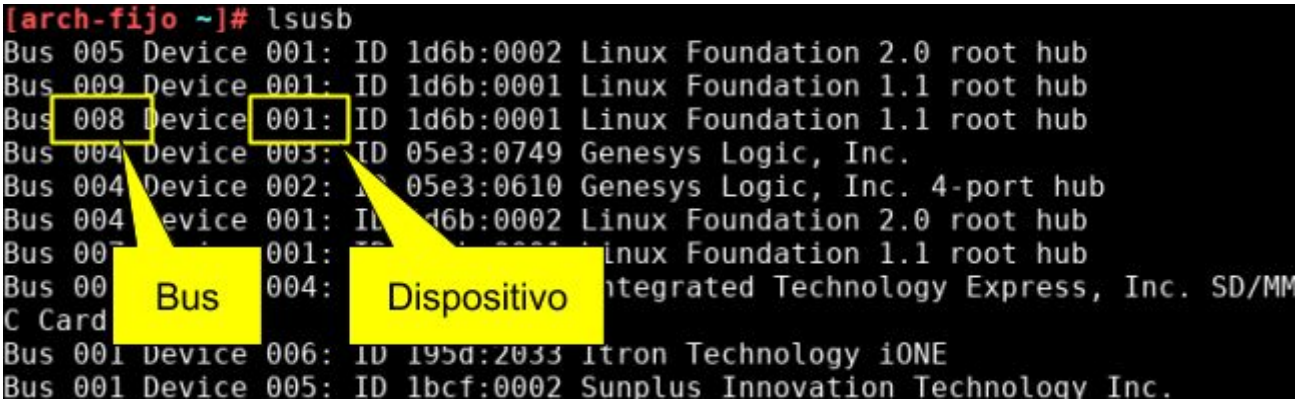
En alguna sección se aprecia la subordinación de unos dispositivos usb respecto a otros



Ejercicio.

1. Toma un par de concentradores usb y conéctalos de forma que uno de ellos esté conectado al computador y otro al primer concentrador.
2. Conecta un ratón al primer concentrador conectado al computador.
3. Mediante el comando lsusb descubre y documenta el hardware que has añadido.
4. Ahora conecta el ratón al segundo concentrador
5. Vuelve a descubrir el mapa de dispositivos usb conectados.

El comando lsusb puede detallar mucha información de un dispositivo, pero hay que indicarle qué dispositivo, indicando su bus y el número de dispositivo mostrado en la salida simple



Ahora usamos la opción -v (verbose) para obtener mucha información de un dispositivo y la opción -s para indicar qué dispositivo concreto:

```
lsusb -s 008:001 -v
```

(la salida es muy larga para pegarla aquí toda)

el comando usb puede filtrar por vendedor o dispositivo, hay una lista de vendedores y dispositivos

Correspondencia entre lsusb y /dev

El comando lsusb devuelve descripciones de dispositivo como la siguiente:

```
$ lsusb
Bus 002 Device 003: ID 0fe9:9010 DVICO
```

Cabe preguntarse qué fichero /dev está el directorio correspondiente a este dispositivo. El dispositivo está conectado al bus usb... que es un bus! Luego la ruta es

```
/dev/bus/usb/002/003
```

lsblk

lsblk lista los dispositivos de bloques (Típicamente discos duros externos o internos y DVD)

hwinfo

Este comando no viene en todas las distribuciones y quizá hay que instalarlo manualmente.

lshw

usb-devices

dmidecode

DMI es "Digital Media Interface", un conjunto de normas que establecen cómo se conectan, reconocen los chips más importantes de una placa base. Estas normas son aprovechadas por el fabricante de la placa base para instalar un programa en la BIOS, que al arrancar, reconoce los dispositivos y rellena una estructura de datos con la información sobre el hardware existente.

Con este comando se obtiene información de dicha estructura de datos sobre todos los principales elementos de la placa base, cpu, cachés, memoria, northbridge y southbridge. En resumidas cuentas: este comando "pregunta" a la bios por el hardware básico conectado a la placa.

biosdecode

biosdecode descifra la información que la bios tiene sobre el sistema. Esta información es únicamente interesante para el sistema operativo en el momento de arrancar.

dmesg

dmesg muestra los mensajes que el núcleo emite para informar de cambios o hechos relevantes ocurridos y que se almacenan en una memoria intermedia circular. Estos mensajes pueden ser de diversa naturaleza, como fallos en aplicaciones, que nada tienen que ver con el hardware. Pero, evidentemente, todos los eventos de hardware, como inserciones o extracciones de dispositivo, pasan por este almacén de mensajes.

Propuestas de ejercicios:

Reconocimiento de los ficheros de arranque:

Objetivo: Reconoce y documenta la configuración del arranque de tu sistema. Ejecuta los siguientes pasos.

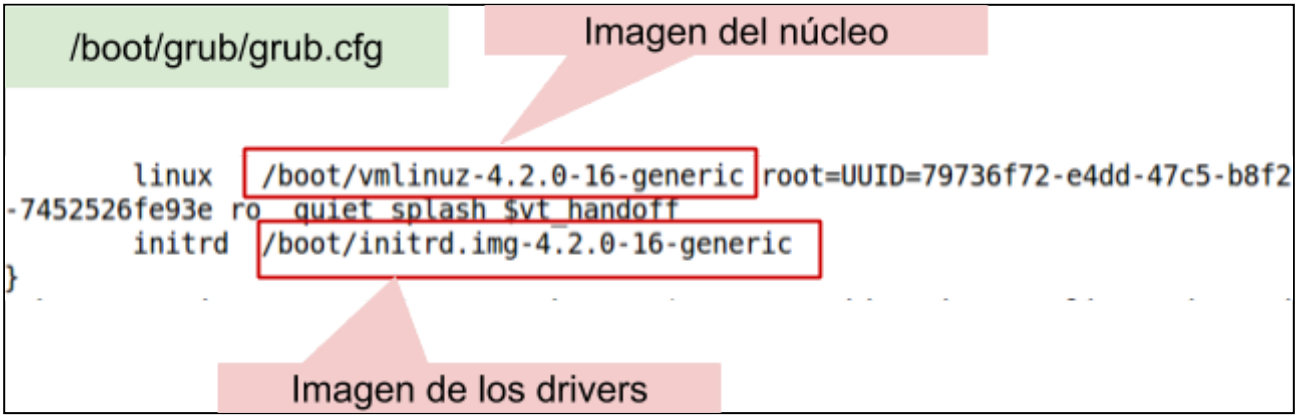
1. Localiza el fichero donde está configurado el arranque para grub. Probablemente sea /boot/grub/grub.cfg
2. Examina el fichero, verás que contiene muchísimas líneas y que apenas se entiende. Trata de localizar las secciones donde se especifican las diversas opciones que grub muestra al iniciarse para que el usuario decida qué arrancar (linux, windows, etc)

   Cada línea es una sección denominada "`menuentry`" que contiene un bloque encerrado entre llaves. A continuación tienes un ejemplo de lo que debes encontrar

```
menuentry 'Void GNU/Linux, with Linux 4.19.23_1 (recovery mode)' --class void
--class gnu-linux --class gnu --class os $menuentry_id_option
'gnulinux-4.19.23_1-recovery-c337a037-6b4f-4541-9189-18954413a705' {
load_video
```

```
set gfxpayload=keep
insmod gzio
insmod part_msdos
insmod ext2
set root='hd0,msdos2'
if [ x$feature_platform_search_hint = xy ]; then
      search --no-floppy --fs-uuid --set=root     --hint-bios=hd0,msdos2
      --hint-efi=hd0,msdos2 --hint-baremetal=ahci0,msdos2
      c337a037-6b4f-4541-9189-18954413a705
else
      search --no-floppy --fs-uuid --set=root c337a037-6b4f-4541-9189-18954413a705
fi
echo   'Loading Linux 4.19.23_1 ...'
linux   /boot/vmlinuz-4.19.23_1 root=UUID=c337a037-6b4f-4541-9189-18954413a705 ro
single
echo   'Loading initial ramdisk ...'
initrd  /boot/initramfs-4.19.23_1.img
}
```

3. En el bloque anterior, hay unas líneas importantes que se muestran en la siguiente captura:



4. Localiza las versión de imagen del núcleo e initrd en tu caso y verifica la existencia de dichos archivos en la ruta indicada

```
lliurex@HiDi-Tec:~$ cd /boot/
lliurex@HiDi-Tec:/boot$ ls
abi-3.19.0-15-generic          memtest86+.bin
abi-4.2.0-16-generic           memtest86+.elf
config-3.19.0-15-generic       memtest86+_multiboot.bin
config-4.2.0-16-generic        System.map-3.19.0-15-generic
grub                           System.map-4.2.0-16-generic
initrd.img-3.19.0-15-generic   vmlinuz-3.19.0-15-generic
initrd.img-4.2.0-16-generic    vmlinuz-4.2.0-16-generic
lliurex@HiDi-Tec:/boot$
```

En esta captura se observan varios fichero vmlinuz asociados a sus initrd.img correspondientes.

Descubrir las carpetas /sys asociadas a un usb insertado

Objetivo: Descubrir fiablemente qué carpetas y ficheros se crean al insertar un dispositivo usb. Para ello, recurrimos a una función que tiene el núcleo de linux llamada inotify, que informa de cambios en el sistema de ficheros. Mediante el uso de un comando, monitorizamos los cambios que ocurren en cualquier subcarpeta de /sys (/sys/devices/) y así descubrimos los cambios que desencadena la inserción del usb. Esto sólo es un ejercicio demostrativo sin casi utilidad real, puesto que lo mismo que vamos a hacer puede hacerse con el comando

```
udevadm monitor
```

Posteriormente vamos a obtener información sobre el fabricante del dispositivo insertado.

Pasos:

Instalamos la utilidad que nos permite utilizar inotify desde la línea de comandos bash

```
apt-get install inotify-watch (xbps-install inotify-watch)
```

Seguidamente lanzamos

```
inotifywatch –r /sys/devices/
```

El comando bloquea el terminal a la espera de eventos y entonces insertamos un usb. Deberían aparecer varias líneas al cabo de unos segundos. El dispositivo usb, en ocasiones requiere unos segundos de negociación hardware e inicialización.

A los segundos, pulsamos ctrl + c, paramos el comando y esto provoca que el comando escriba la secuencia de eventos en la pantalla. A continuación tienes una parte de una captura de un experimento similar:

```
[nacho@void sys]$ inotifywatch -r /sys/devices/ 2>&1
Establishing watches...
Finished establishing watches, now collecting statistics.
^Ctotal  access  close_nowrite  open  filename
98    34    32           32    /sys/devices/pci0000:00/0000:00:12.2/usb1/
36    12    12           12    /sys/devices/pci0000:00/0000:00:13.2/usb2/
36    12    12           12    /sys/devices/pci0000:00/0000:00:13.0/usb5/
36    12    12           12    /sys/devices/pci0000:00/0000:00:12.2/usb1/1-3/
36    12    12           12    /sys/devices/pci0000:00/0000:00:12.0/usb3/
36    12    12           12    /sys/devices/pci0000:00/0000:00:13.1/usb6/6-1/
36    12    12           12    /sys/devices/pci0000:00/0000:00:13.1/usb6/
36    12    12           12    /sys/devices/pci0000:00/0000:00:12.1/usb4/
6     2     2            2     /sys/devices/pci0000:00/0000:00:13.2/usb2/2-0:1.0/
6     2     2            2     /sys/devices/pci0000:00/0000:00:13.0/usb5/5-0:1.0/
...
```

Ahora, tenemos una secuencia de eventos, pero no todos los eventos corresponden al dispositivo insertado, muchos son eventos en los buses que llegan hasta él. Elegimos la línea más larga, que corresponde con el dispositivo.

6 2 2 2 /sys/devices/pci0000:00/0000:00:13.1/usb6/6-1/6-1:1.0/

(Alternativamente se podría realizar la monitorización al extraer el dispositivo usb, se obtendrían menos eventos y por tanto información más clara, pero no es lo natural)

Ahora, examinamos el modalias que se encuentra en la carpeta elegida, para descubrir el vendor. Nos vamos a

https://pci-ids.ucw.cz/

y buscamos el código del fabricante

Ampliación

Repite el experimento metiendo el usb en el mismo puerto del ordenador, pero anota cada vez (o guarda la salida de inotify) las rutas de las carpetas creadas. ¿Se crean y destruyen las mismas carpetas?

Esta vez, utiliza el proceso con el comando

```
sudo inotifywatch -v -r /sys/
```

Práctica recuperación de arranque en linux. ¡Muy útil!

Objetivo: En este ejercicio se va a corromper o destruir el sistema de arranque de un computador con linux, para posteriormente arreglarlo. Necesitas un computador con un linux instalado y que tenga capacidad de arrancar de un CD o USB con una imagen de linux. Es preferible realizarlo en una máquina virtual y no arriesgarse a que algo vaya mal con un computador real.
[Nota para profesores: cada paso está explicado con una imagen, pero el alumno debería recibir una explicación mayor enlazándola con conceptos sobre linux en general hasta que en un futuro se realice la expliación aquí]

Romper cosas

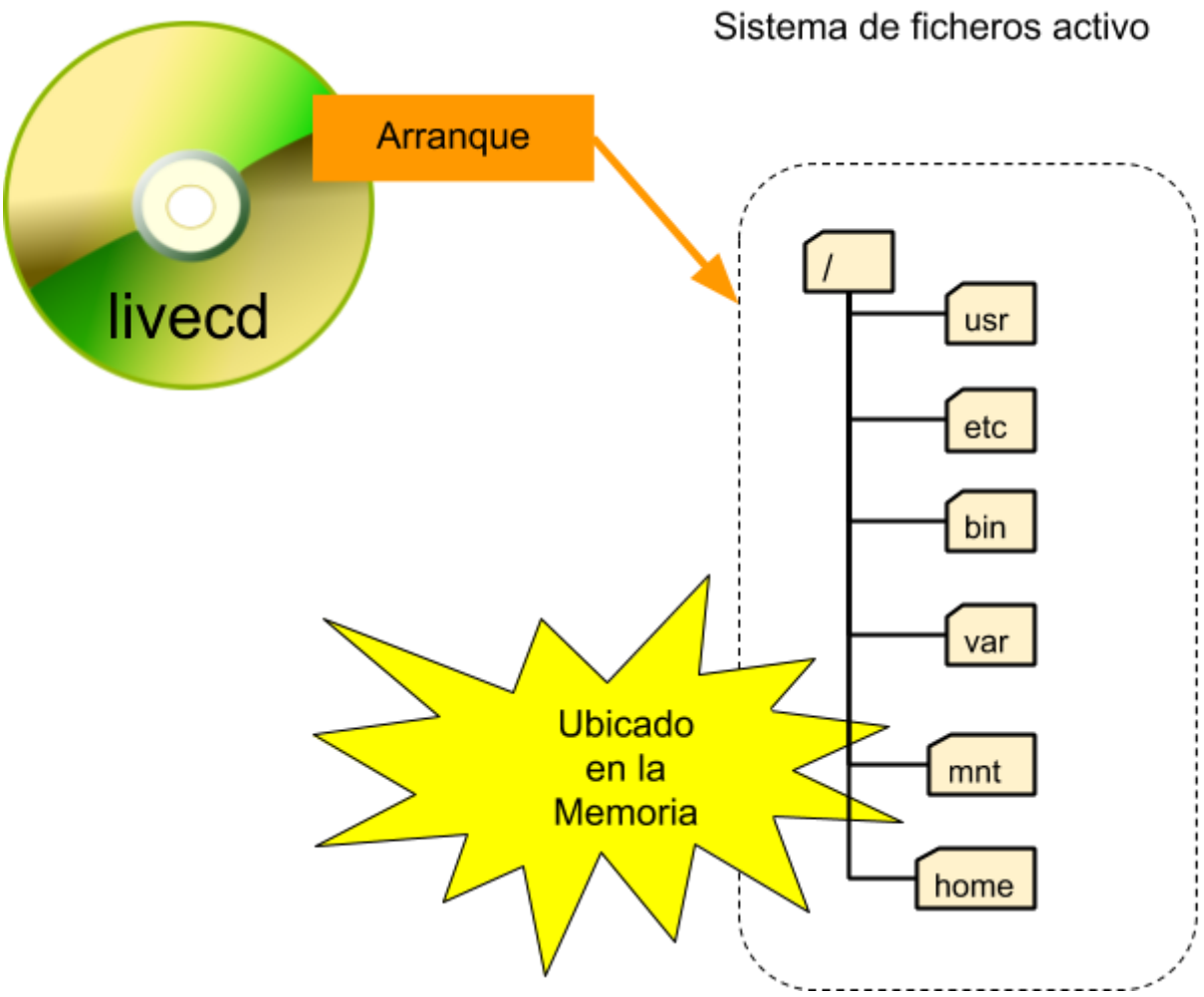
1. Arrancar con un live cd
2. Averiguar la versión del núcleo cat /proc/version
3. Machacar el fichero initrd con contenido basuraf
   cat ficheroBasura >**/boot/initrd-XXXXXXX** (averiguado en paso anterior)
   cat /boot/XXXXXXX > **/boot/initrd-XXXXXXX**

dd **if=/dev/random** of=**/boot/initrd_XXXXXX** bs=2048 count=100
<explicar por qué es mejor que un rm>

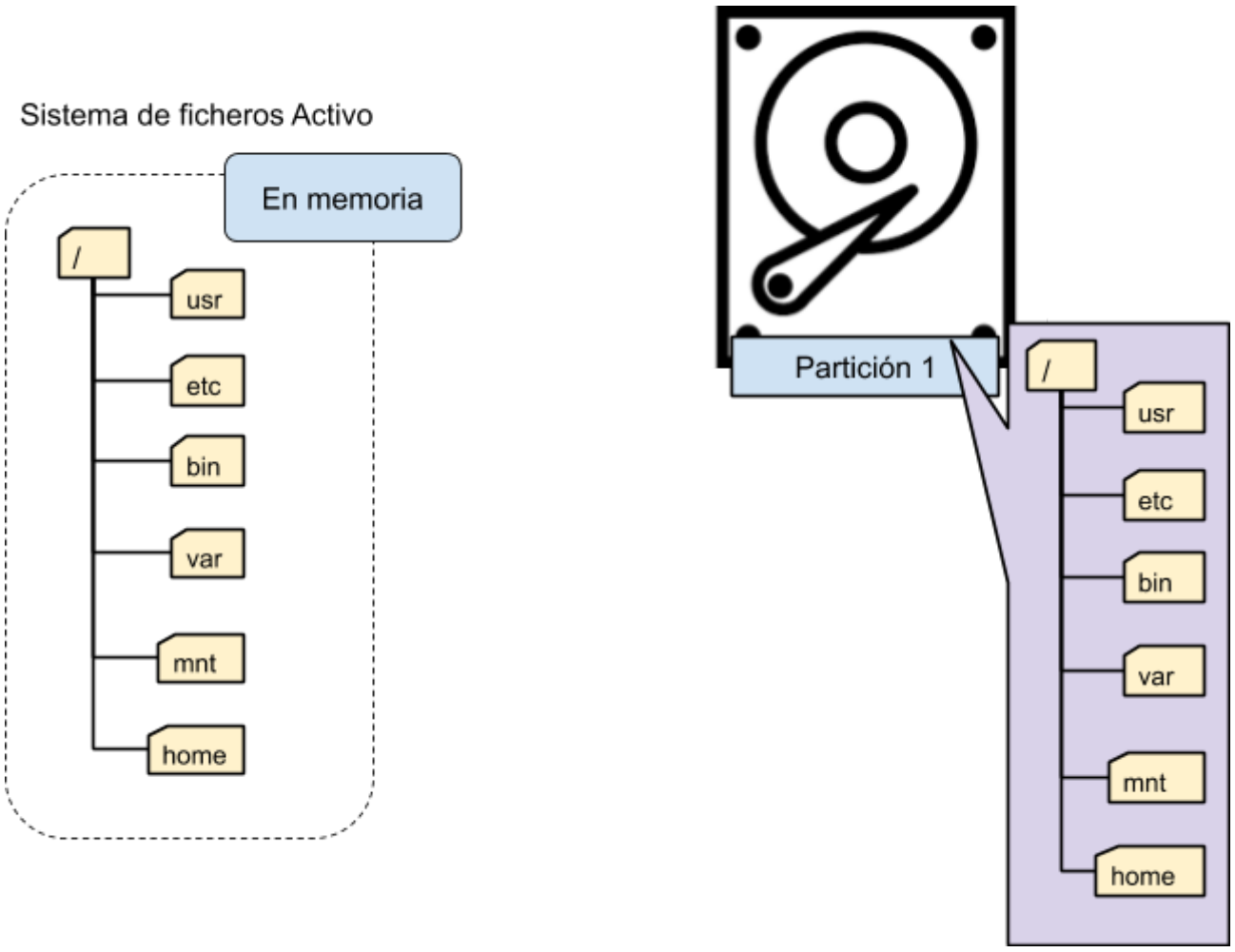
4. Reiniciar (Ya hemos hecho mucho daño) reboot

Arreglar cosas:

1. Arrancar con un live CD (idealmente la misma distro que hay instalada)

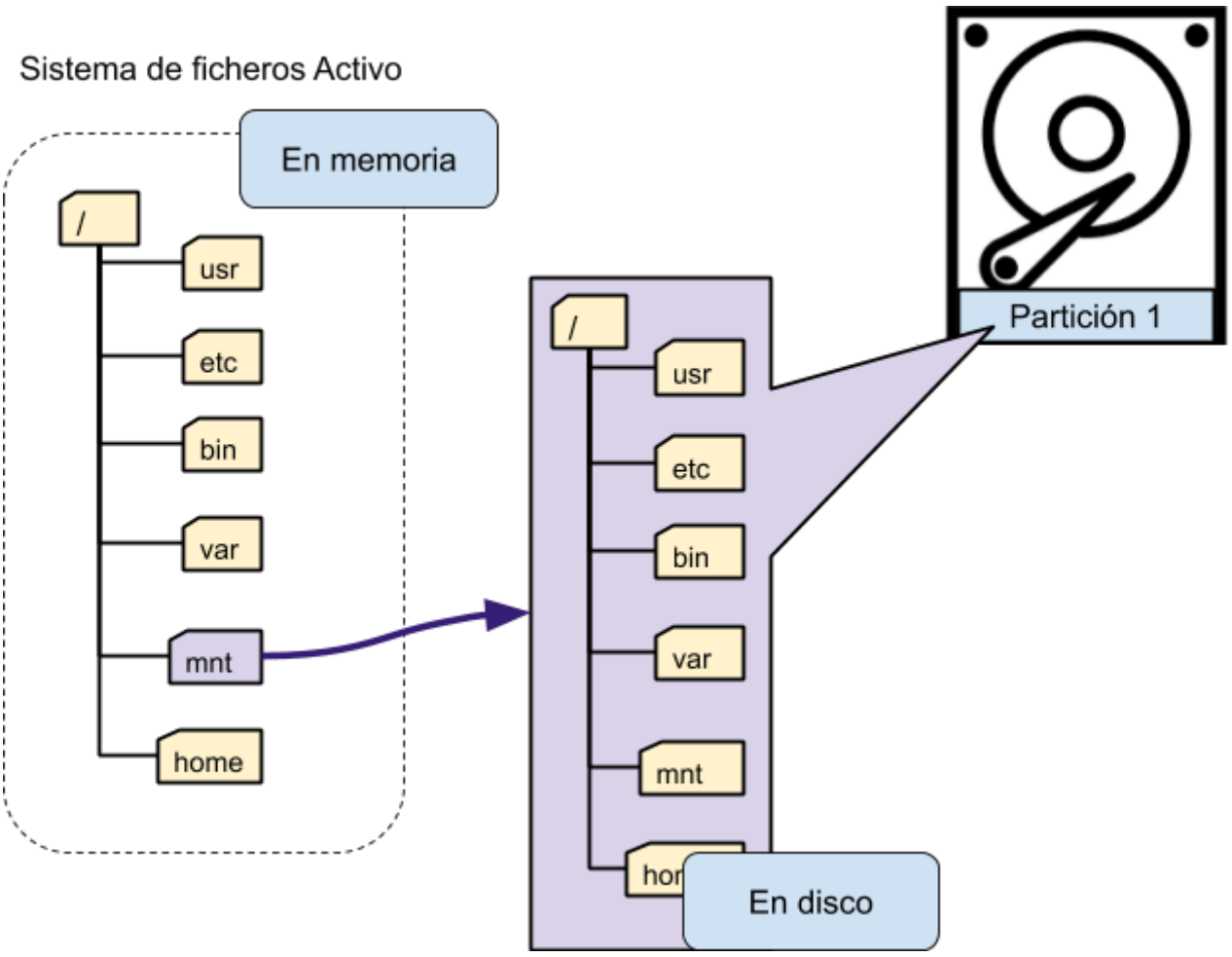


2. Averiguar la partición estropeada fdisk -l



3. Montar la partición estropeada

sudo mount /dev/sda5 /mnt.

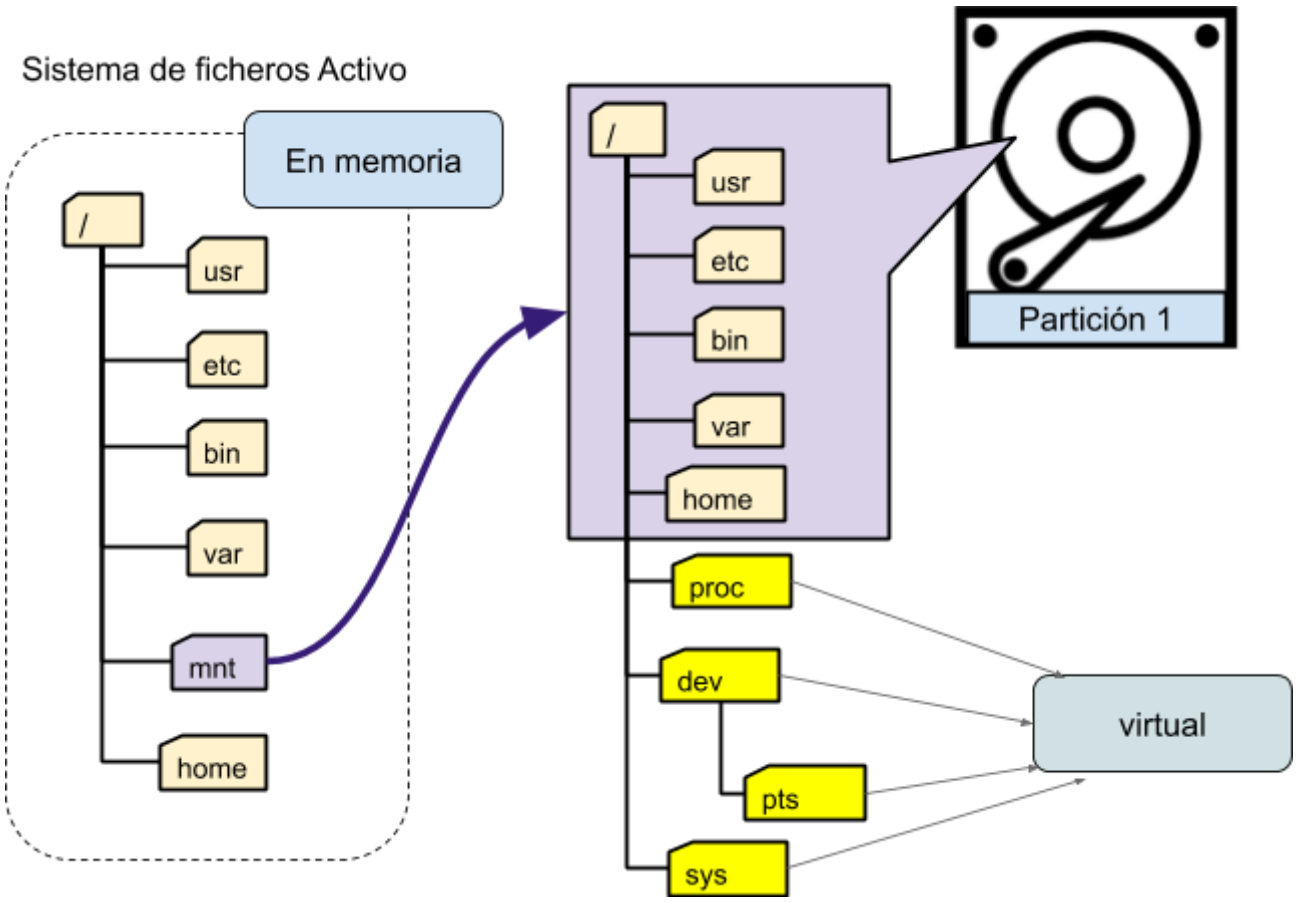


La partición está accesible y se podría recuperar datos de ella, pero aquí vamos a reparar la instalación.

4. Montar el resto de carpetas virtuales ("proc", "sys", "dev", " dev/pts") colgando de la carpeta montada

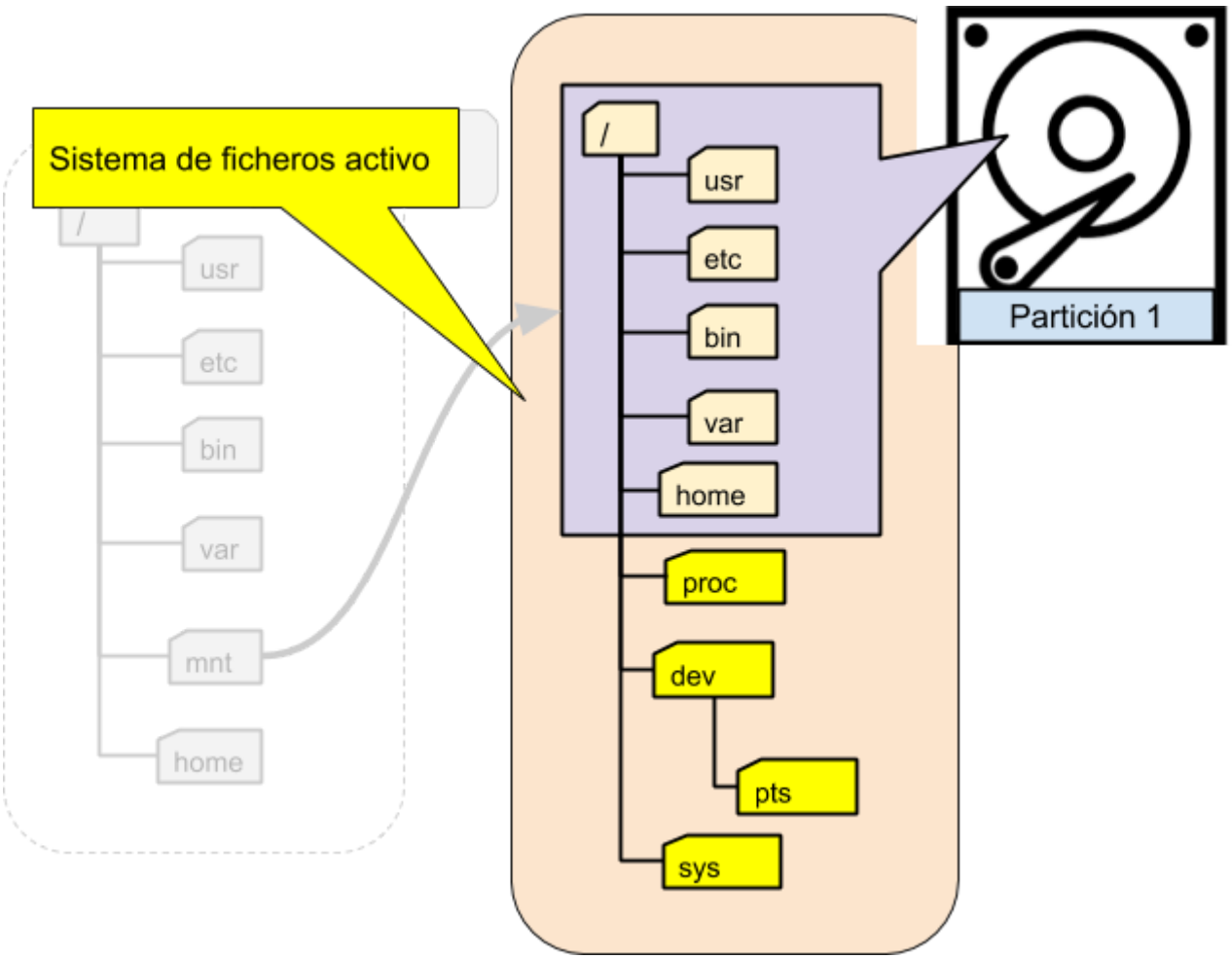
```
sudo mount --bind /dev /mnt/dev
sudo mount --bind /dev/pts  /mnt/dev/pts
sudo mount --bind /proc /mnt/proc
sudo mount --bind /sys  /mnt/sys
```

Nota: Esto no crea carpetas en la partición montada. Tan sólo hace visibles esas carpetas virtuales en la ubicación indicada



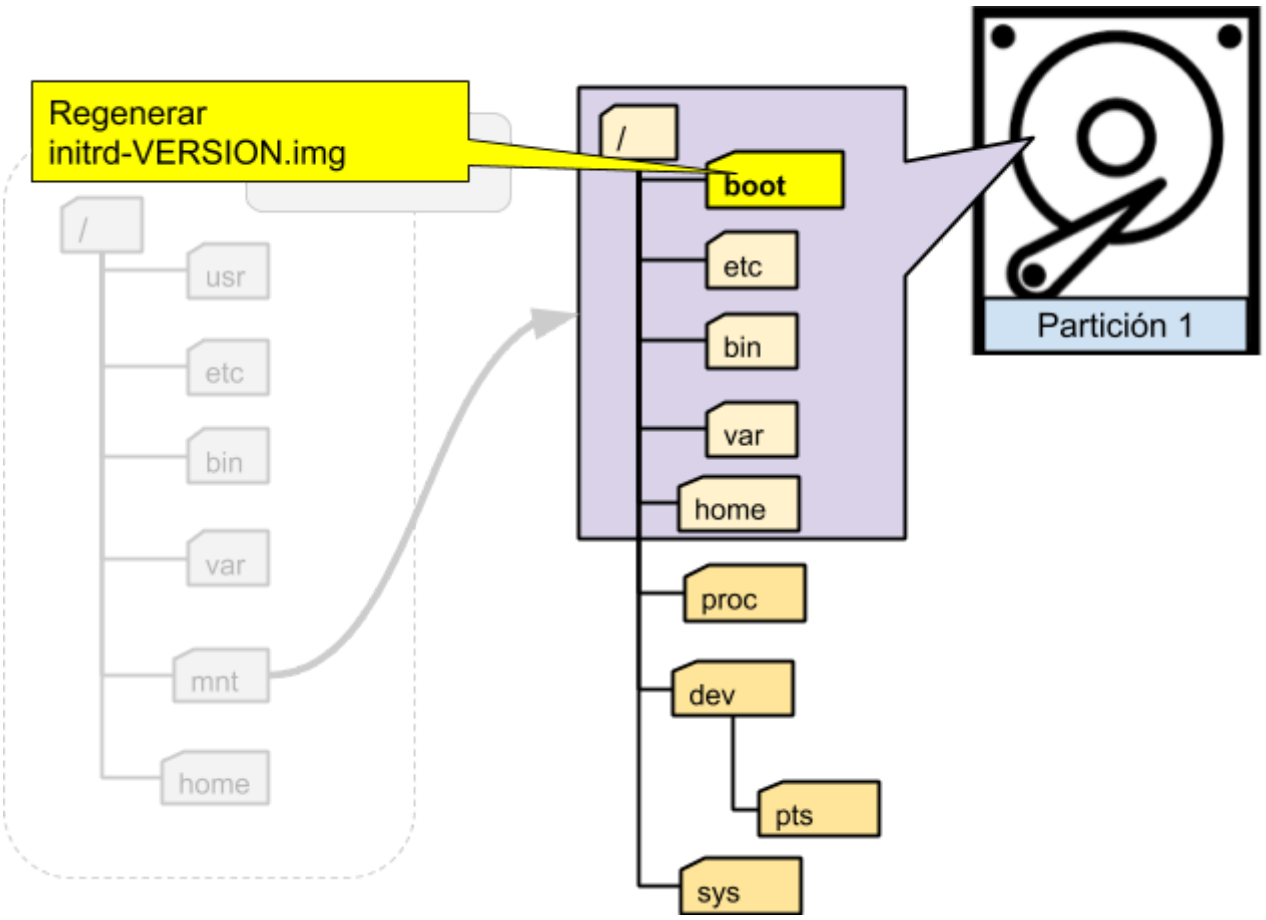
5. Cambiar el sistema de fichero Raíz del terminal de forma que la Raíz ahora será el directorio recién montado

```
$ sudo chroot /mnt
```



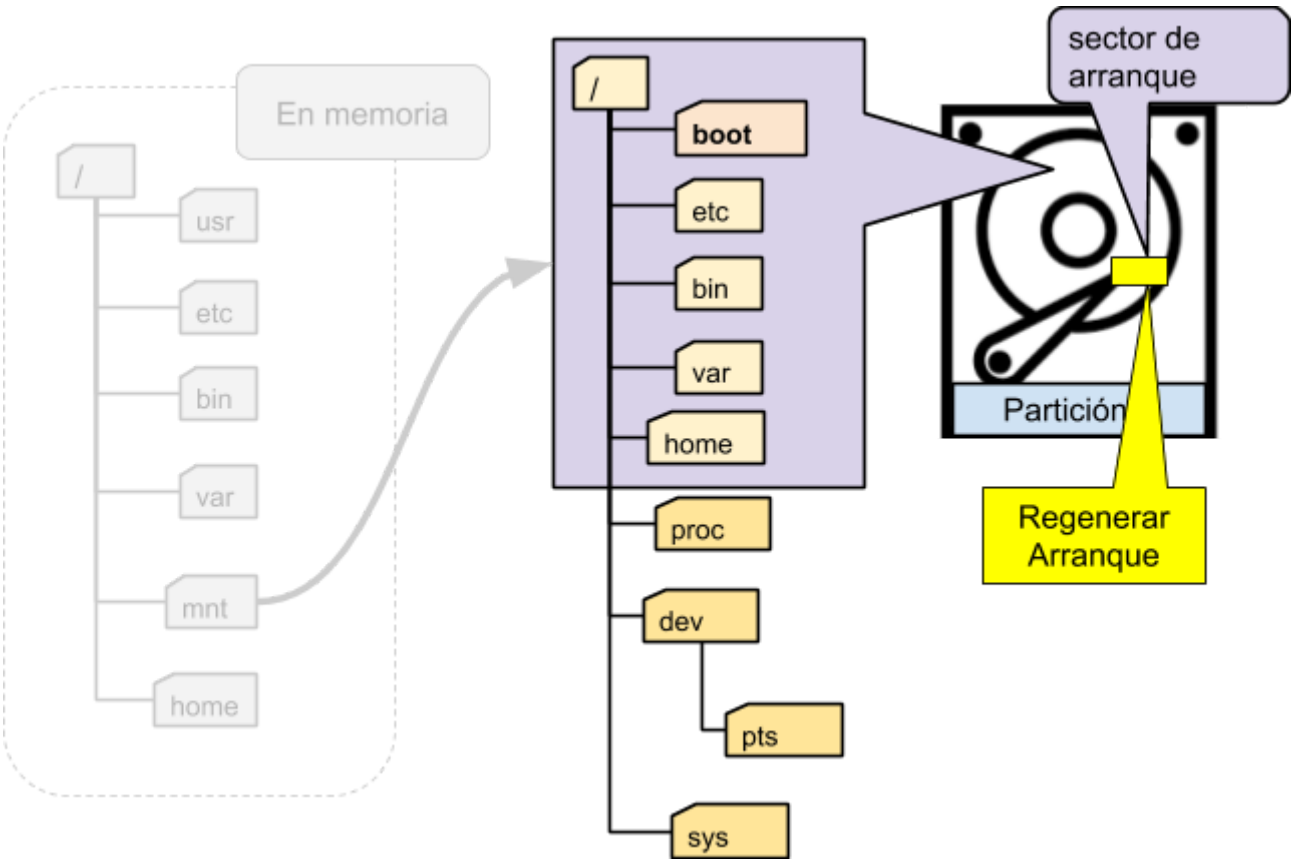
6. Regenerar el fichero initrd acorde al núcleo usado

```
update-initramfs -u -v
```



7. Regenerar el arranque (el arranque necesita prepararse para cargar el nuevo fichero exáctamente desde la ubicación del disco donde está, por ello hace falta regenerar el arranque, sólo con crear el fichero otra vez no sirve)

```
grub-install --recheck /dev/sda
update-grub
```



El trabajo ya está hecho, en el próximo reinicio todo debería funcionar bien. Sin embargo, es conveniente desmontar adecuadamente la partición y por ello hay que salir ordenadamente

8. Salir del "chroot" para poder reiniciar y, reiniciar

```
exit
reboot
```

9. Pasar la factura al cliente (5 minutos)

Reactivar un usb "extraído" por el usuario

Descripción del problema:

Al insertar una memoria usb, se desencadenan una serie de eventos que, para el usuario terminan por mostrar un icono de unidad nueva en el navegador de archivos:

<poner captura del expolorador de archivos centrándose en el icono de la unidad usb insertada>

Después de trabajar con la unidad usb, el usuario puede elegir la opción de "extraer"

(posiblemente hay otra opción de "desmontar", nos centraremos en "extraer").

<poner captura del expolrador de archivos centrándose en la opción de extraer>

Lo que esta opción provoca en última instancia, es que el bus usb "olvide" que está conectado el dispositivo usb. Una vez "extraído" el dispositivo desde el explorador de archivos, ya no es posible "conectarlo" si no se extrae físicamente y se vuelve a insertar.

Como se ha dicho, el usb es desconectado y todo el sistema operativo se olvida del dispositivo. Nuestro objetivo es reactivarlo.

La parte interesante de este ejercicio es que esta desactivación de dispositivo ocurren otras veces en otros dispositivos y es interesante conocer el procedimiento para reactivarlo

Pasos:

Lo ideal sería dar el primer paso después de haber extraído el dispositivo desde el menú del explorador de archivos. Lamentablemente, <hasta donde yo sé> la solución tiene una fase de ir dando palos de ciego. Por ello, en esta actividad, vamos a hacer algo antes de extraer el dispositivo que nos abrevie el procedimiento

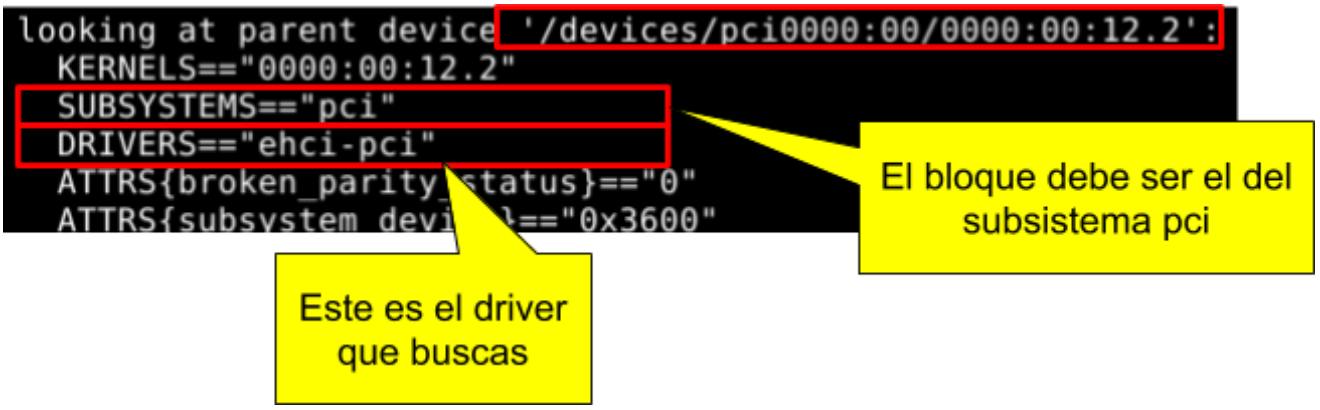
Después de meter el usb hay que conocer información sobre él. En especial debemos averiguar uno de los drivers implicados en el uso del dispositivo. Los buses usb han ido evolucionando a lo largo de los años y existen diversos drivers asociados a ellos que gestionan el puente pci-usb. Del dispositivo usb, sólo podemos saber mediante el comando `mount` o `lsblk` qué dispositivo /dev es. Suponiendo que el usb es /dev/sdc, ejecuta el siguiente comando:

```
udevadm info -a -n /dev/sdc
```

En caso de ser un teclado, podrías hacer algo así:

```
udevadm info -a -n /dev/input/by-id/usb-046a_0011-event-kbd
```

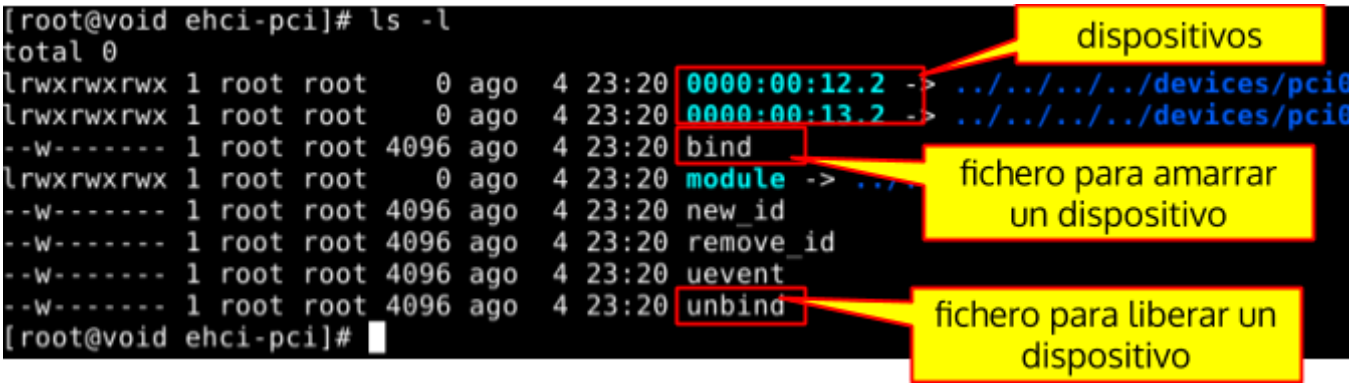
Aparecerá abundante información (que te debería ser familiar después de estudiar el tema). De entre toda hemos de fijarnos en el driver pci que está gestionando el subsistema donde está el bus asociado. Recuerda, el bus usb está colgando del bus pci. Aquí no interesa el bus usb u otros usados, sino el bus pci que está implicado. De toda la información que aparece, hacia el final puedes observar las claves DRIVERS y SUBSYSTEMS:



Ahora que tenemos claro el driver, nuestro siguiente paso será encontrar una carpeta dentro de /sys donde continuaremos el trabajo. En la carpeta /sys/bus, aparecen carpetas correspondientes a los buses del sistema. En nuestro caso nos hemos de fijar en la carpeta pci. Dentro de esta carpeta existe otra carpeta llamada drivers. Esta carpeta mantiene información sobre los drivers que están gestionando el bus pci. Cada versión de bus usb (en un sistema puede haber varios buses usb de diferentes versiones) tiene un driver distinto. Como ya has obtenido el driver anteriormente hay que entrar en la carpeta del driver adecuado. en el caso de la memoria usb la carpeta podría ser:

```
/sys/bus/pci/drivers/ehci-pci/
```

Dentro de esta carpeta aparecen enlaces simbólicos a todos los dispositivos que están gestionados por este driver. También aparecen dos ficheros importantes "bind" y "unbind" (en inglés "amarrar" y "liberar").



Necesitas saber cuál de los dispositivos que aparecen ahí es el que necesitas reactivar. Por suerte, con el comando udevadm info anterior en el que has mostrado toda la información, aparece la ruta completa dentro de /sys. Tan sólo debes observar el componente de la ruta que indica qué dispositivo es. Por ejemplo, la siguiente línea (incompleta) se obtiene con ese comando:

```
/devices/pci0000:00/0000:00:12.2/usb1/1-1/1-1:1.0/host8/target8:0:0/8:0...
```

La parte subrayada es la que delata de qué dispositivo se trata. Ahora ya podemos activar y desactivar a voluntad el dispositivo:

Activar el dispositivo (desde el directorio /sys/bus/pci/drivers/ehci-pci/):

```
echo 0000:00:12.2 > bind
```

Desactivar el dispositivo:

```
echo 0000:00:12.2 > unbind
```

Bastará con hacer una activación o desactivación+activación.

El siguiente script activa y desactiva todos los dispositivos usb:

```
for i in /sys/bus/pci/drivers/[eo]hci-pci/*:*; do
  [ -e "$i" ] || continue
  echo "${i##*/}" > "${i%/*}/unbind"
  echo "${i##*/}" > "${i%/*}/bind"
done
```

Activar un servicio a la inserción de un usb

Requisitos: Conocer el sistema de inicio "systemd" y especialmente la idea de "unit" como configuración de los servicios

Objetivo: Reaccionar a la inserción de una memoria USB lanzando un servicio (servicio gestionado por systemd).

Pasos previos

Lubuntu 19.04, tomamos un usb ("sunstri"), lo insertamos, lo monta automáticamente y procedemos a descubrir sus atributos. La forma fácil es

udevadm info -a -n /dev/sdd

/dev/sdd es el disco usb insertado. (la otra forma es usando rutas /sys asociadas al dispositivo, hay que descubrirlas , por ejemplo con udevadm monitor al insertarlo), por ejemplo:

udevadm info -a -p /devices/pci0000:00/0000:00:13.2/usb2/2-4/2-4:1.0/host9/target9:0:0/9:0:0:0/block/sdd

De todos los atributos que aparecen , vamos a fijarnos en :

ATTRS{serial}=52ABFAF1

Procedimiento

Hay que preparar dos cosas:

1. Un servicio systemd en base a un fichero .unit
2. Un script o un programa que será el servicio
3. Una regla de udev

Unit:

La unit (creo) que en principio sólo será un fichero "usb.service" ubicado en /lib/systemd/system

```
[Unit]
Description=Unit para reaccionar a la insercion de un usb
#bindsTo=activated unit de la que dependo, si para aquella, para ésta.
#after= así esperamos que el dispositivo esté totalmente en marcha para
iniciar esto

[Service]
ExecStart=/bin/bash /usr/bin/nacho-script.sh

[Install]
#WantedBy=aso.target
```

El script que representará el servicio real es:

```
#/bin/bash

# in the future, the file name will depend on the device name
rm /tmp/nacho.output
a=1;
while [ true ] ; do
      sleep 2;
      echo "a=$a" >>/tmp/nacho.output
      let a=a+1
done
```

Regla:

Esta regla udev siguiente funcionó totalmente, incluso arrancó el servicio (la parte de RUN+ está para evidenciar el disparo de la regla independientemente de que el servicio vía systemd funcione"

```
ATTR{serial}=="52ABFAF1",TAG+="systemd",
ENV{SYSTEMD_WANTS}="usb.service",RUN+="/usr/bin/touch /tmp/activo"
```

(La parte de RUN+ es por depuración, se puede eliminar)pruebas

Lo siguiente no parece desencadenar las reglas udev, pero sí que avisa de que hay un dispositivo coincidente

```
udevadm trigger -v  -a "serial"="52ABFAF1"
```

Activar un template a la inserción de un usb

La idea es similar al ejercicio anterior, pero ahora vamos a activar un servicio al que se le pasará algún parámetro originado en el dispositivo insertado. Por ejemplo, si se inserta un usb que se manifiesta en /dev/sdd, entonces "/dev/sdd" será recibido por el script lanzado

Hay que preparar igualmente:

1. Un template systemd, que es como un servicio, pero el nombre es especial, lleva una @ al final del nombre del servicio como veremos
2. Una regla udev que es similar a la del anterior ejemplo, pero activa un servicio basado en el template

template

La principal diferencia del template respecto a un servicio es el nombre. Nuestro template va a ser un fichero llamado

template-usb@.service

Esa @ indica que es un template y no una unit de servicio normal, y además marca que entre esa @ y el "." se insertará alguna cadena variable cuando se instancie un servicio concreto

Escribamos ahora el template:

```
[Unit]
Description=Template para reaccionar a la insercion de un usb
#bindsTo=activated unit de la que dependo, si para aquella, para ésta.
```

```
#after= así esperamos que el dispositivo esté totalmente en marcha para
iniciar esto

[Service]
ExecStart=/bin/bash /usr/bin/template-usb.sh %I

[Install]
#WantedBy=aso.target
```

Observa que el template es una unit .service de lo más normal. La diferencia en este caso es el especificador o parámetro %I que ves usado dentro de la unit y que se le pasa al script o programa que implementa el servicio como argumento. Existen más especificadores, pero %I suele ser el habitual.

Este template podrá ser instanciado para activar varios servicios, todos réplicas de esta unit. Cada de uno de ellos puede recibir un nombre en la instanciación diferente. Así, para este template, por ejemplo, algo podría iniciar el servicio:

```
systemctl start template-usb@1000.service
```

systemd, al observar un "@" en el nombre de la unit del comando anterior, busca el template

```
template-usb@.service
```

Y lo lee para activar el servicio. Pero el "1000" es un dato que puede ser consultado dentro del template. El especificador %I es reemplazado por ese 1000. observa que el %I es usado como argumento en la línea ExecStart= y por tanto pasado como argumento a nuestro script, que ahora puede aprovecharlo:

```
#/bin/bash

rm /tmp/template-usb.output
echo "parametro recibido: $1" > template_activo
a=1;
while [ true ] ; do
      sleep 2;
      echo "a=$a - $1" >>/tmp/template-usb.output
      let a=a+1
done
```

Regla udev

La regla udev es la que ahora causa el inicio del servicio basado en el template. Tiene cierta libertad para darle un nombre al servicio y éste nombre puede cambiar en diferentes disparos de la regla, basándose igualmente en especificadores que se obtienen directamente

de parámetros del hardware insertado o del núcleo. Usaremos y explicaremos la siguiente regla (que copiaremos en /etc/udev/rules.d/20-template-usb.rules) :

```
ATTR{serial}=="52ABFAF1",
TAG+="systemd",
ENV{SYSTEMD_WANTS}="template-usb@%k.service",
```

Observa el nombre de la unit asociada, es el template, pero con un %k añadido después de "@", este %k concretamente es el nombre del dispositivo para el núcleo (por ejemplo "sdd" en un disco duro). Esa variable "%k" cambia a cada disparo de la regla y por tanto se invoca un servicio nuevo cada vez

Vincular un servicio a un dispositivo con bindTo

Se puede lograr algo similar a lo anterior sin intervenir para nada en udev. La idea es que systemd crea una unit de tipo device por cada dispositivo. Estas units pueden ser la base de otras units hechas por nosotros. De forma que nuestra unit llevará las siguientes cláusulas

```
[Unit]
Description=Automatically rip inserted DVDs
After=dev-cdrom.device
BindsTo=dev-cdrom.device
Requisite=dev-cdrom.device
```

- Con "After" indicamos que cuando se active la unit "dev-cdrom.device" deberemos activar ésta.
- Con Requisite, indicamos que hasta que no esté totalmente activa la unit "dev-cdrom.device" no empezaremos con ésta (tiene sentido si hay que tener disponible la información del dispositivo).
- Con "BindsTo" indicamos que la vida de esta unit está ligada a la de "dev-cdrom.device" y que al detenerse aquella, se detendrá nuestra unit.

Averiguar usando la web, la marca y modelo de la tarjeta gráfica

<se debe proporcionar la clase de dispositivo, con ello se sabe el directorio /sys/class donde buscar, se llega hasta el modalias y se obtiene el código donde mrar en la web.

Averiguar también el comando que obtiene la misma información>

aço está per fer.

Prohibir o permitir explícitamente dispositivos

http://www.irongeek.com/i.php?page=security/plug-and-prey-malicious-usb-devices#3.2_Locking_down_Linux_using_UDEV

Idea: desactivar el dispositivo escribiendo un 0 en el fichero authorized de la carpeta correspondiente de /sys

Idea: en una regla udev, la variable $DEVPATH nos indica la ruta /sys del dispositivo

Combinando ambas ideas, aparece la siguiente regla como candidata:

```
ACTION=="add",
ATTR{serial}=="078606B90DD3",
RUN+="/bin/sh -c 'echo 1 >/sys$DEVPATH/authorized'"
```

(SI una regla no cabe en una línea, será escrita AQUÍ en varias, pero en la práctica real del ejercicio deberá seguir siendo una sola línea)

Problema: En esa ruta no hay un fichero authorized muchas veces!!

Solución: hay que ir subiendo en la jerarquía de carpetas a partir de la que udev nos entrega, buscando el primer fichero authorized que se encuentre y escribir un 0

La regla debe llamar a un script y pasarle la ruta::

```
ATTRS{name}=="HID 046a:0011",
RUN+="/bin/bash –c '/usr/bin/desautorizar.sh $DEVPATH'"
```

Y el script que proporciona la solución (ojo esta versión muestra mucho mensaje para ver cómo funciona):

```
#!/bin/bash

carpeta=$1
carpeta="/sys${carpeta}"

#Empecem

encontrado=false
rm -f /tmp/regla.log

while [ $encontrado == false ] ; do
     echo "buscando en $carpeta/" >> /tmp/regla.log
```

```
      if [ -f "$carpeta/authorized" ] ; then
            echo "encontrado fichero authorized en " >> /tmp/regla.log
            echo "$carpeta/authorized" >> /tmp/regla.log
            echo 0 > $carpeta/authorized
            encontrado=true
      fi
      carpeta=${carpeta%/*}
      if [ -z "$carpeta" ] || [ "$carpeta" == "/" ] ; then
                  echo "authorized no  encontrado" >> /tmp/regla.log
                  exit
      fi
done
```

Al ejecutar este script se busca hacia arriba una carpeta con fichero authorized. Observa la captura (la primera línea está recortada para mostrar mejor la salida)

```
[root@void ~]# ./desautorizar.sh /devices/pci0000:00/0000:00:12.0/usb3/3-1/3-1:1.0/0003:046A:0011.000D/input/
buscando en /sys/devices/pci0000:00/0000:00:12.0/usb3/3-1/3-1:1.0/0003:046A:0011.000D/input/input30/event18/
buscando en /sys/devices/pci0000:00/0000:00:12.0/usb3/3-1/3-1:1.0/0003:046A:0011.000D/input/input30/
buscando en /sys/devices/pci0000:00/0000:00:12.0/usb3/3-1/3-1:1.0/0003:046A:0011.000D/input/
buscando en /sys/devices/pci0000:00/0000:00:12.0/usb3/3-1/3-1:1.0/0003:046A:0011.000D/
buscando en /sys/devices/pci0000:00/0000:00:12.0/usb3/3-1/3-1:1.0/
encontrado fichero authorized en
/sys/devices/pci0000:00/0000:00:12.0/usb3/3-1/3-1:1.0/authorized
echo 0 > /sys/devices/pci0000:00/0000:00:12.0/usb3/3-1/3-1:1.0/authorized
[root@void ~]#
```

¡Probado en void linux!

Ejercicio no solucionado ni desarrollado, <pendiente>

Ejercicio.
Configura un sistema que reaccione a la inserción de una memoria USB concreta de forma que sólo con ser insertada realice una copia de los ficheros de un directorio de un usuario a dicha memoria USB.

idea, usando los comandos anteriores, descubre la forma de identificar a una partición específica (pista uuid) y haz que el script (que no la regla) sea capaz de reaccionar exactamente a dicha partición. usa udevadm info --env para obtener la información total de la partición cuando la insertas

Existen diversos comandos (dmidecode, usbXXXX, lspci, hwinfo, lshw, lscpu,cpuid, udevadm,usbview,lsusb, lsblk, usb-devices etc.) que también muestran información sobre el hardware. Algunos acceden al bus directamente y otros leen lo existente en /sys (o /proc ?)

https://chrisjean.com/how-to-find-hardware-devices-in-ubuntu-with-lshw/

http://www.reactivated.net/writing_udev_rules.html#testing

Ejercicio

That won't work. RUN is for short program invocations only and udev enforces this. Also, it would have been killed by systemd anyway when parent udev exits.

The clean way to do it using modern systemd/udev infrastructure.

1. Create systemd service that starts you backup script on request. Do not put it in background! I.e. remove { ... } & wrapper. Otherwise system will believe your script finished as soon as it is started:

Code:

bor@opensuse:~> cat > /etc/systemd/system/backup\@.service << EOF
[Unit]
Description=Backup to USB Flash Disk
BindsTo=dev-%i.device

[Service]
Type=simple
ExecStart=/usr/local/bin/backupUSB.sh %I
EOF
bor@opensuse:~> systemctl daemon-reload
This is template unit (it has "@" in its name). Later you will dynamically instantiate template by calling it with device name. You refer to device name using %i and %I placeholders (latter is capital "i" not small "L"). BindsTo ensure service is stopped when device in unplugged.

2. Create udev rule to start service when device in plugged in.

Code:

bor@opensuse:~> cat > /etc/udev/rules.d/99-backup.rules << EOF
```
KERNEL=="sd?1", ACTION=="add", SUBSYSTEMS=="scsi",
ATTRS{vendor}=="JetFlash", ATTRS{model}=="Transcend 4GB   ",
RUN+="/usr/bin/systemctl --no-block start backup@%k.service"
EOF
```
bor@opensuse:~> udevadm control --reload
Now when you plug in USB stick service backup@sdX1.service should be started. You can check status using "systemctl status backup@sdX1.service".

nombres complejos de dispositivos en bbdd

Escenario, varios alumnos usan diferentes usb en el computador que comparten como grupo, para distinguirlos bien, van a hacer un fichero de texto donde anotarán en cada línea un atributo del dispositivo y el nombre. Por ejemploo

Anexos

Claves modalias

```
"v", VENDOR,
"p", PRODUCT,
"dl", DEV_LO,
"dh", DEV_HI,
"dc", DEV_CLASS,
"dsc",DEV_SUBCLASS,
"dp",DEV_PROTOCOL,
"ic",INT_CLASS,
"isc",INT_SUBCLASS,
"ip",INT_PROTOCOL,
```

claves udev

$kernel, %k

The kernel name for this device.

$number, %n

The kernel number for this device. For example, 'sda3' has kernel number of '3'

$devpath, %p

The devpath of the device.

$id, %b

The name of the device matched while searching the devpath upwards for SUBSYSTEMS, KERNELS, DRIVERS and ATTRS.

$driver

The driver name of the device matched while searching the devpath upwards for

$attr{file}, %s{file}

The value of a sysfs attribute found at the device, where all keys of the rule have matched. If the matching device does not have such an attribute, follow the chain of parent devices and use the value of the first attribute that matches. If the attribute is a symlink, the last element of the symlink target is returned as the value.

$env{key}, %E{key}

A device property value.

$major, %M

The kernel major number for the device.

$minor, %m

The kernel minor number for the device.

$result, %c

The string returned by the external program requested with PROGRAM. A single part of the string, separated by a space character may be selected by specifying the part number as an attribute: %c{N}. If the number is followed by the '+' char this part plus all remaining parts of the result string are substituted: %c{N+}

$parent, %P

The node name of the parent device.

$name

The current name of the device node. If not changed by a rule, it is the name of the kernel device.

$links

The current list of symlinks, separated by a space character. The value is only set if an earlier rule assigned a value, or during a remove events.

$root, %r

The udev_root value.

$sys, %S

The sysfs mount point.

$tempnode, %N

The name of a created temporary device node to provide access to the device from a external program before the real node is created.

%%

The '%' character itself.

$$

The '$' character itself.

Enlaces

https://frankpzh.wordpress.com/2011/04/16/kmod-udev-and-modprobe/

https://superuser.com/questions/881929/udev-pci-device-rule-is-not-working

El comando que se usa para añadir un módulo al núcleo es modprobe. Podemos decir que udev es el que invoca a modprobe para cargar el driver adecuado en el núcleo.

(interesante) Lo hace udev-> https://unix.stackexchange.com/questions/127005/how-does-init-determine-which-devices-to-modprobe

documentación interna del núcleo:

https://www.kernel.org/doc/Documentation/driver-model/platform.txt

https://unix.stackexchange.com/questions/330186/where-does-modprobe-load-a-driver-that-udev-requests