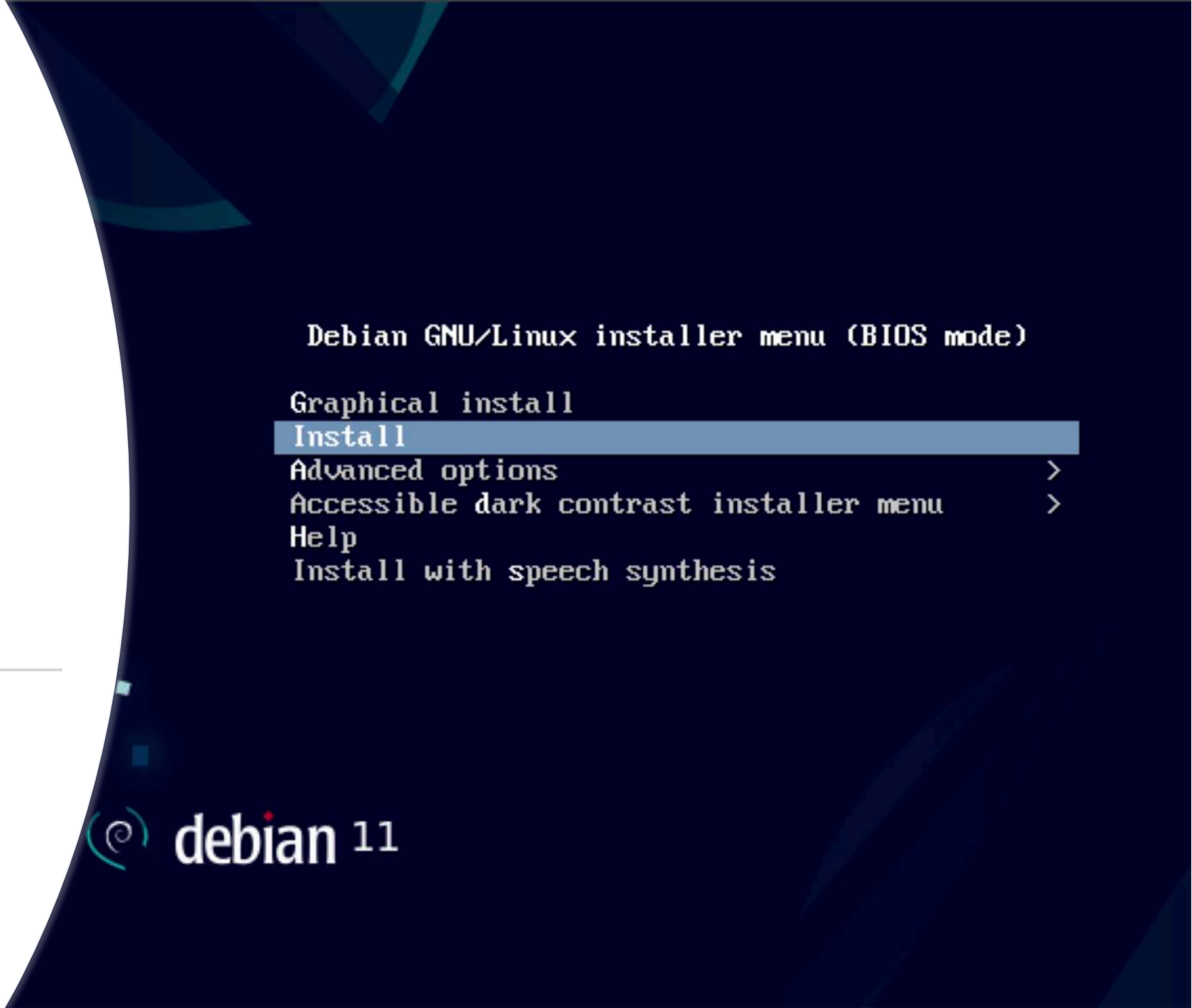
Sistemas Operativos
I – Introducción (Prácticas)

Dr. Mikel Emaldi Manrique

Universidad de Deusto – Facultad de Ingeniería

1. Instalación del SO



Construyendo un SO

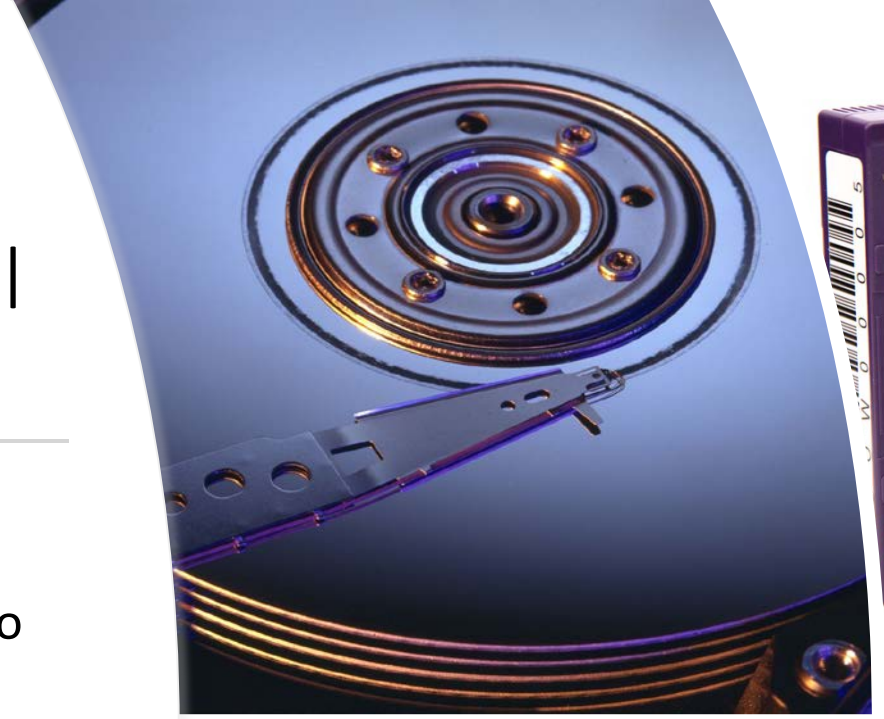
- Aunque es posible crear un SO para una máquina específica, lo óptimo es que sea compatible con el mayor número de periféricos.
- Pasos para crear un SO:
    1. Escribir el código fuente (u obtener uno escrito previamente).
    2. Configurar el SO para el sistema en el que se va a ejecutar.
    3. Compilar el SO.
    4. Instalar el SO.
    5. Arrancar el ordenador con el nuevo SO.
- El administrador del sistema podría seleccionar los módulos que deseamos instalar, incluso modificar el núcleo según sus necesidades.
- También podemos funcionar con máquinas virtuales.

Arrancando el SO

- Pero, ¿cómo sabe el sistema dónde está el núcleo o qué núcleo debe arrancar?
- Arranque del sistema (booting):
    1. Un pequeño programa, normalmente ubicado al principio del disco duro, localiza el núcleo: gestor de arranque, **bootstrap program** o **boot loader**.
        - En detalle, primero se carga un boot loader conocido como la **BIOS (basic input/output system)**, alojada en la placa base.
        - Que, a su vez, carga un segundo boot loader, ubicado en el **boot block**.
        - Algunos sistemas modernos han sustituido la carga a través de la BIOS por **UEFI** (Unified Extensible Firmware Interface).
    2. El núcleo se carga en memoria principal y se inicia.
    3. El núcleo inicializa el hardware.
    4. Se monta el sistema de ficheros raíz.
- GRUB: gestor de arranque open-source.

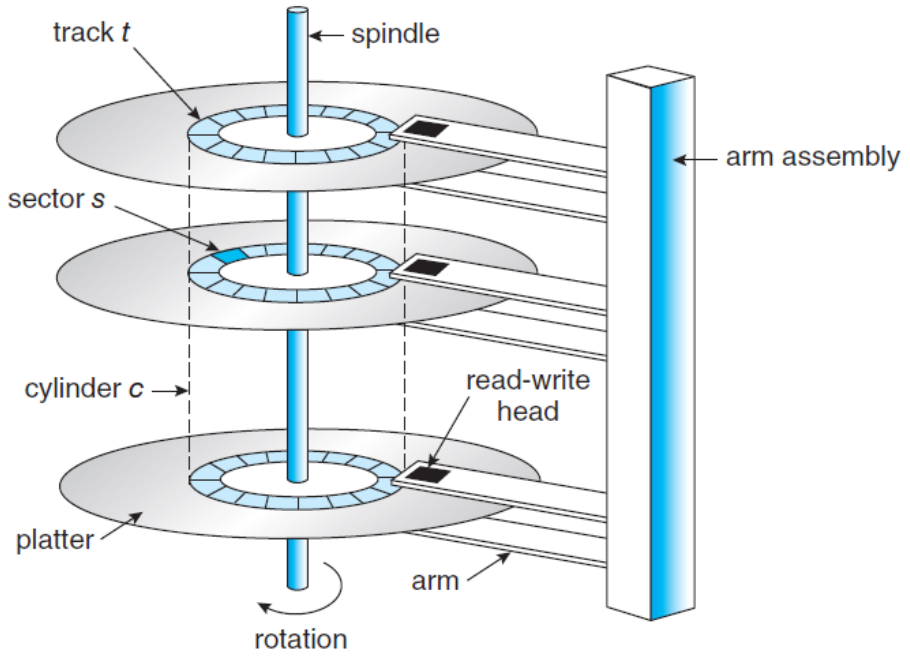
Almacenamiento no-volátil

- En los computadores modernos, tenemos mayormente dos tipos de almacenamiento no-volátil:
  - **Discos duros** (HDD, Hard Disk Drives): dispositivos de almacenamiento electromecánicos.
  - **Memoria no-volátil** (NVM, Non-Volatile Memory): dispositivos de almacenamiento basados en memorias de semiconductor.
- Otros dispositivos como cinta magnética, discos ópticos o almacenamiento cloud.



Discos duros (HDD)

- Formados por **platos** de forma circular, cuyas caras están cubiertas por un material magnético
- Los **cabezales de lectura/escritura** están unidos a los **brazos**, que se mueven de manera simultánea.
- Cada plato está divido en **pistas circulares**, que a su vez, están divididas en **sectores**.
- El conjunto de pistas accesibles en una determinada posición de los brazos, se denomina **cilindro**.



Dispositivos de memoria no-volátil

- Dispositivos eléctricos/electrónicos.
- Controlador + memoria flash NAND basada en semiconductores.
  - Memoria flash NAND: desarrollada por Kioxia en 1987.
- **Discos duros de estado sólido** (Solid-State Disk, SSD).
- Dispositivos USB (pendrive), discos NVME (Non-Volatile Memory Express).
- Más rápidos que los HDD ya que no hay dispositivos mecánicos que tienen que moverse (entre otras características).
- Consumen menos energía.
- Son más caros.

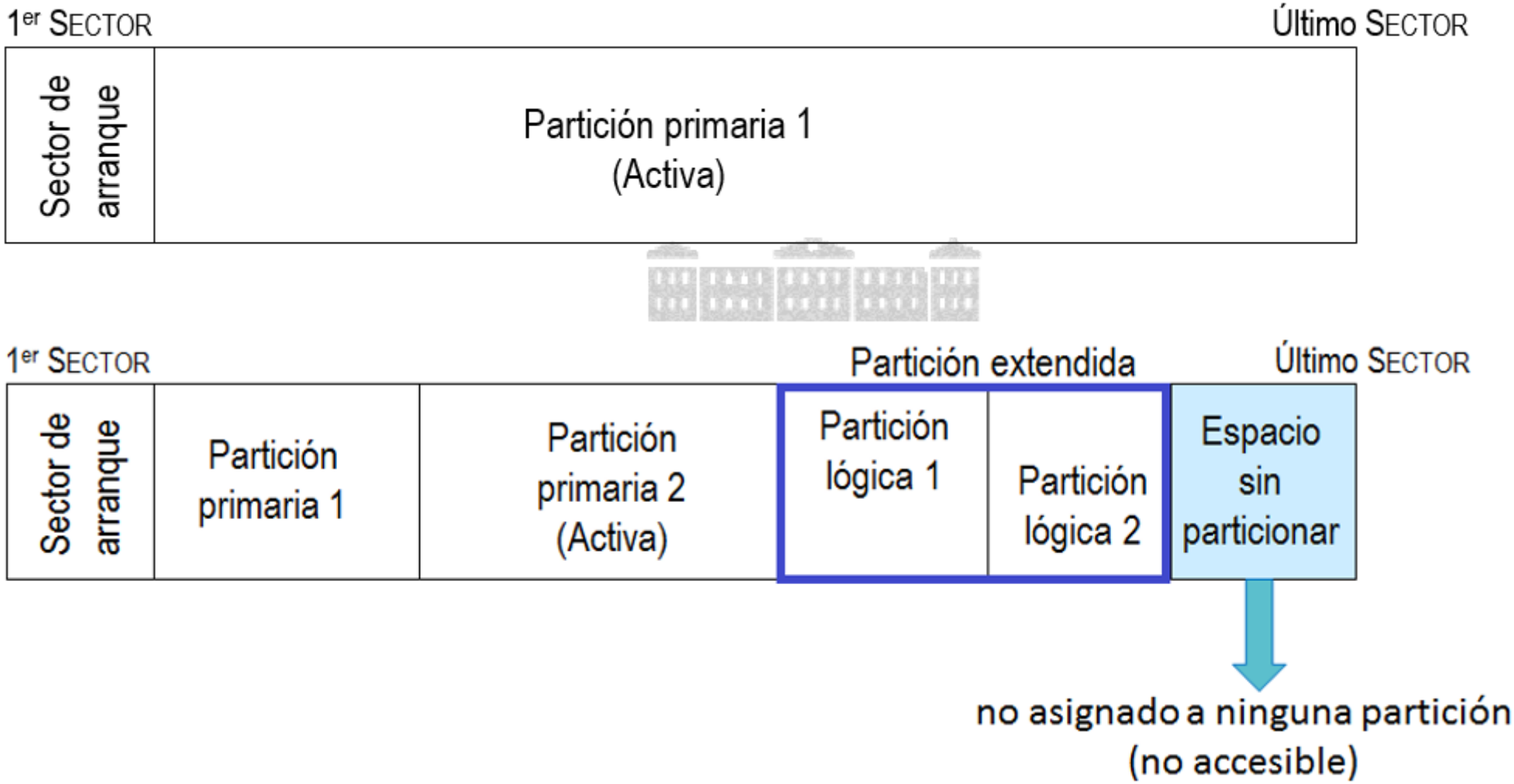
Formateo físico del dispositivo

- Un dispositivo nuevo es como un “lienzo en blanco”:
  - Platos de material magnético.
  - Un conjunto de celdas de almacenamiento de semiconductores.
- Antes de almacenar datos:
  - Dividir el disco en sectores que el controlador (driver) pueda leer o escribir.
  - Inicializar las páginas NVM y la FTL (Flash Translation Layer).
- Este proceso es denominado **formateo físico** o **formateo de bajo nivel**.
- Muchos discos duros traen este formateo realizado de fábrica.

Particiones (I)

- Antes de que el SO pueda utilizar un dispositivo de almacenamiento, necesita **crear sus propias estructuras de datos**.
- Tres pasos: particionado, creación del volumen y formateo lógico.
- **Partición**: grupo de bloques o páginas.
- El SO puede tratar cada partición como si fuese un **dispositivo independiente**.
- Por ejemplo:
  - Una partición para el código ejecutable del SO.
  - Otra para el área de intercambio (swap).
  - Otra para el los ficheros de usuario.
- La información de particionado se escribe en una zona específica del disco.
- **Montar**: hacer que el sistema de ficheros esté disponible para el SO.

Particiones (II)



Particiones (III)

- Particiones **primarias** y **extendidas**.
- Particiones primarias:
  - Necesario que exista al menos una partición primaria para arrancar el SO.
    - Para instalar el **boot loader**.
  - **Máximo de 4 particiones primarias** en un disco.
    - EL **MBR (Master Boot Record)** sólo tiene 4 punteros a particiones.
- Partición extendida:
  - Es una partición primaria especial.
  - Puede contener N particiones lógicas.
  - **Máximo 1 partición extendida**.

Volúmenes y sistemas de ficheros

- Volúmenes:
  - El segundo paso para que el SO pueda utilizar el disco, es la **creación del volumen**.
  - Es la representación lógica en la cual se puede crear el **sistema de ficheros**.
  - Muchas veces se crea de manera implícita al crear el sistema de ficheros.
- Formateo lógico:
  - Proceso para crear el **sistema de ficheros**.
  - Tabla de localización de ficheros (mapeo entre ficheros lógicos y físicos), espacio libre disponible...
  - Diferentes sistemas de ficheros: NTFS, ext4, FAT, exFAT, APFS (Apple File System), MacOS Extended...

Sector de arranque (boot record/block)

- Cuando se enciende el ordenador, necesitamos un programa inicial para arrancar el SO: programa **Bootstrap, boot loader o gestor de arranque**.
- Un primer gestor instalado en la BIOS o en el firmware de la memoria flash NMV.
  - Arranca un segundo gestor más complejo.
- El gestor de arranque completo, está almacenado en el **sector de arranque o boot block**.
- Linux: grub2.

Secuencia de arranque de Linux

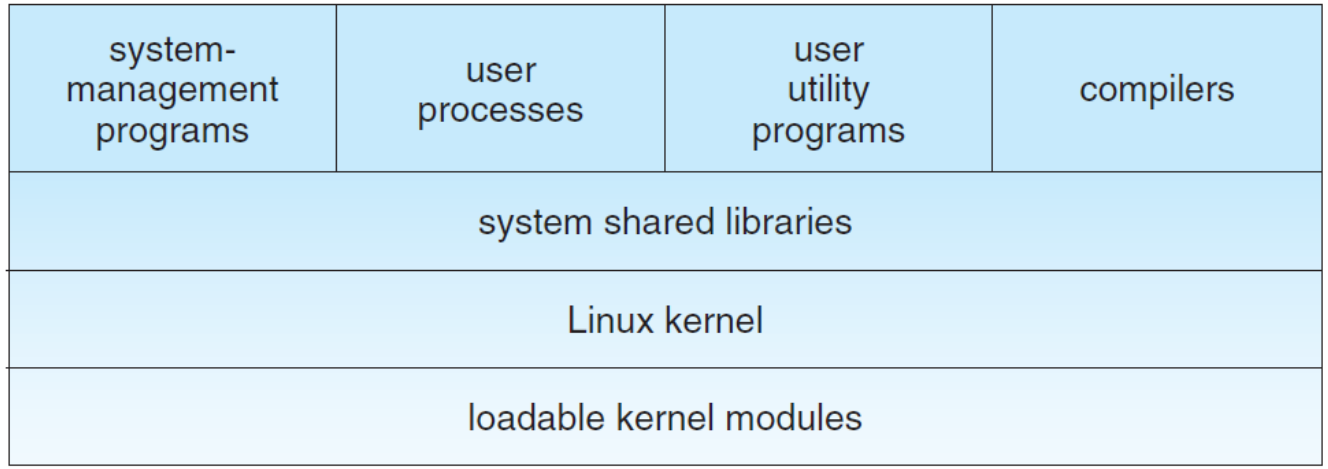
1. Al encender el ordenador se inicia la **secuencia de arranque**.
2. La BIOS ejecuta la **secuencia POST** (Power on Self Test). Inicializa y comprueba el hardware.
3. El gestor de **arranque de la BIOS** selecciona el dispositivo de arranque y lo arranca. Para el HDD, carga el **MBR** en memoria.
   - UEFI es una alternativa más moderna al arranque BIOS.
4. El MBR selecciona la **partición activa** y carga el PBR (Partition Boot Record).
5. EL PBR carga la **siguiente secuencia de arranque** (segundo programa de arranque) en memoria.
6. El **boot loader o programa de arranque** carga el **núcleo** de Linux y los **drivers** seleccionados en memoria.
7. El kernel inicializa su **estado interno** y otras **estructuras de datos** necesarias para su funcionamiento.
8. Una vez se inicializa el kernel, comienza **la ejecución de procesos**. El proceso **inicial (init o systemd)** continua con la inicialización del sistema.
9. Una de las etapas de la inicialización del sistema es iniciar los mecanismos para permitir la **autenticación de usuarios**.
10. Una vez que el usuario se autentica, se carga su configuración y se inician los **procesos de usuario**.

Arquitectura de Linux (I)

- **Núcleo monolítico** + **módulos** que permiten modificar el comportamiento del núcleo durante la ejecución del SO.
- Núcleo de Linux:
  - Desarrollado desde 0 por la comunidad Linux.
  - Implementa las funciones fundamentales del SO.
- Sistema Linux:
  - Componentes software que no tienen por qué ser exclusivos a Linux.
  - Berkeley's BSD, MIT's X Window System o **proyecto GNU de la Free Software Foundation**.
  - P. ej.: GNU Compiler Collection (gcc), File Transfer Protocol (FTP).
- Distribuciones de Linux:
  - Al principio, era necesario compilar tanto el núcleo como los diferentes componentes y aplicaciones necesarias para ejecutar Linux.
  - Distribuciones: colecciones de paquetes que incluyen mucho más que un sistema Linux básico.
  - Los **gestores de paquetes** fueron su avance más significativo.

Arquitectura de Linux (II)

- Linux está compuesto por tres componentes principales:
  - **Núcleo (kernel)**. Responsable de mantener todas las abstracciones principales del SO, incluyendo la memoria virtual y los procesos.
  - **Librerías del sistema.**
    - Definen una serie de funciones estándar a través de las cuales las aplicaciones pueden interactuar con el núcleo.
    - Implementan mucha de la funcionalidad del SO que no necesita los privilegios del núcleo.
    - **C library (libc)**. Implementa la librería estándar de C y el modo usuario de la interfaz de las llamadas al sistema de Linux.
  - **Utilidades del sistema.** Programas que ejecutan tareas de gestión individuales y especializadas.

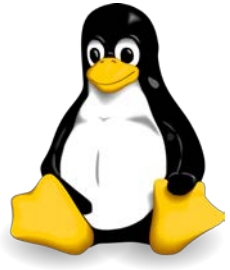


Módulos del núcleo (kernel modules) (I)

- Linux puede **cargar y descargar secciones arbitrarias** del **núcleo** bajo demanda.
- Estos módulos se ejecuten en **modo privilegiado o modo núcleo**.
- En teoría, **no hay ninguna restricción** respecto a lo que un módulo del núcleo puede hacer.
- Por ejemplo, pueden implementar un driver de dispositivo, un sistema de ficheros, o un protocolo de red.
- **Ventajas** de los módulos del núcleo:
  - No hace **falta recompilar todo el núcleo** para añadir una nueva funcionalidad.
  - Pueden desarrollarse módulos que no cumplan la **licencia GPL** bajo la que está distribuido el núcleo de Linux.
  - Permiten configurar un sistema Linux con un **núcleo mínimo**, sin ningún driver de dispositivo que no sea necesario.

Módulos del núcleo (kernel modules) (II)

- Linux incluye los siguientes componentes para habilitar los módulos:
  1. **Sistema de gestión de módulos:** permite que los módulos sean cargados en memoria y se comuniquen con el resto del núcleo.
  2. **Cargador (loader) y descargador (unloader) de módulos:** utilidades de usuario, trabajan con el sistema de gestión de módulos para cargar (o descargar) un módulo a memoria.
  3. **Sistema de registro de drivers:** permiten a los módulos anunciar la existencia de un nuevo driver al resto del núcleo.
  4. **Mecanismo para la resolución de conflictos:** permite que los diferentes drivers de dispositivo reserven recursos hardware y los protege para evitar el uso accidental por parte de otros drivers.

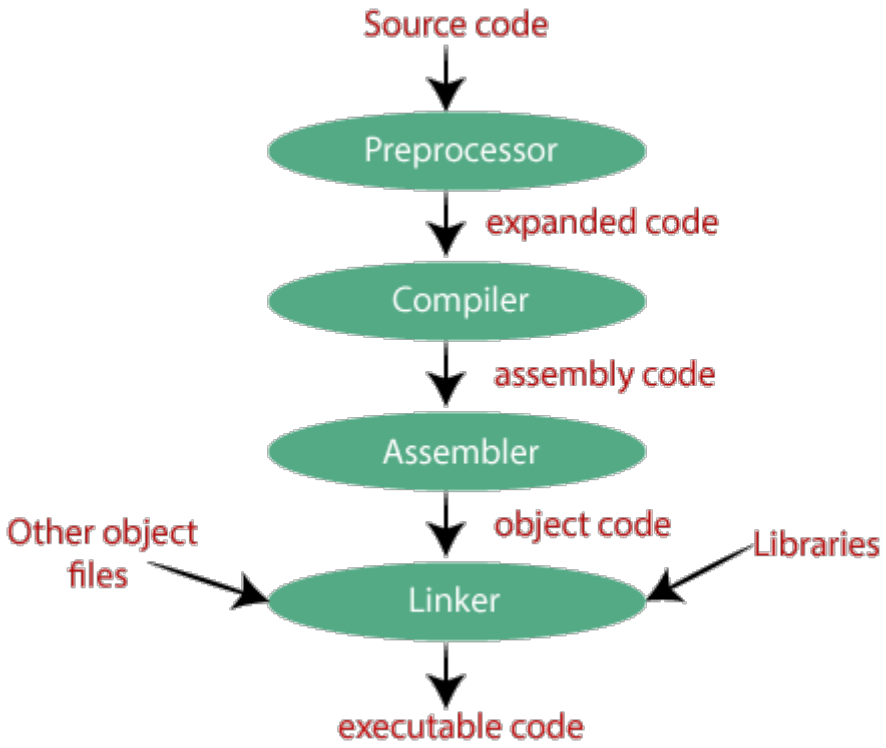


Núcleo de Linux

- Podemos descargar el código del núcleo de Linux de https://kernel.org/
- Algunas de los directorios que podemos encontrar:
  - **arch**: código específico de cada una de las arquitecturas soportadas. Un subdirectorio para cada una de ellas, p. ej., i386 o Alpha.
  - **include**: ficheros “include” (librerías) necesarias para compilar el núcleo.
  - **init**: código de inicialización del núcleo.
  - **mm**: código del gestor de memoria.
  - **drivers**: código de los drivers de dispositivo.
  - **ipc**: código del IPC (inter-process communications).
  - **modules**: directorio donde se almacenan los módulos compilados.
  - **fs**: código del gestor de ficheros. Incluye un subdirectorio por cada sistema de ficheros soportado.
  - **kernel**: código del núcleo.
  - **net**: código de la red del núcleo.
  - **lib**: código de las librerías del kernel.
  - **scripts**: scripts que son utilizados para configurar el kernel.

Compilador de C (gcc) (I)

- **gcc**: GNU Compiler Collection.
- Desarrollado originalmente por Richard Stallman.
- Pasos para generar un ejecutable:
  - **Código fuente**: código que escribe el desarrollador.
  - **Código expandido** o preprocesado: código al que se han añadido todas las librerías necesarias para su compilación
  - **Código ensamblado**: código en lenguaje ensamblador.
  - **Código objeto**: código máquina.
  - **Código ejecutable**: fichero en el que se han linkado los diferentes ficheros objeto que forman nuestro programa, para crear el ejecutable final.

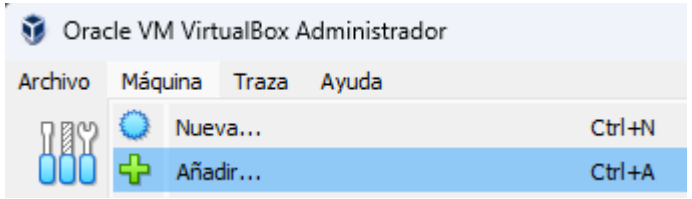


Compilador de C (gcc) (II)

- Podemos para el proceso en cualquiera de los pasos con los siguientes atributos:
  - -e: para tras el preprocesado.
  - -s: para tras la compilación.
  - -c: para tras el ensamblado.
- Ejemplos:
  - Compilar el contenido del archivo “programa.c” sin montarlo:
    - $ cc –c programa.c
  - Compilar el contenido del archivo “programa.c” y crear el ejecutable con nombre por defecto “a.out”:
    - $ cc programa.c
  - Compilar el contenido del archivo “programa.c”, crear el ejecutable y darle nombre:
    - $ cc -o programa programa.c

Máquina Virtual de Debian

- En el aula vamos a utilizar una MV de Debian para hacer las prácticas.
- Para crear la máquina :
  1. Descargamos la MV de ALUD.
  2. Descomprimimos su contenido a un directorio.
  3. En VirtualBox pulsamos en Máquina → Añadir.
  4. Navegamos al directorio dónde hemos descomprimido la MV y seleccionamos el fichero “SO Debian.vbox”.
  5. Una vez importada la máquina pulsamos en “Iniciar”.
  6. El usuario y contraseña de la máquina son debian/debian.



- También nos podemos conectar a la máquina a través de un túnel ssh:
  - ssh -p 2222 debian@localhost