Unidad 4: EL SISTEMA OPERATIVO

BLOQUE II – Sistemas, aplicaciones y personas

CONTENIDOS

1. Introducción.
2. Diseño de sistemas operativos y kernel.
3. Llamadas al sistema.
4. Interfaces de usuario.
5. El proceso de arranque
6. Virtualización.

1. Introducción

- Los sistemas operativos modernos proporcionan el entorno de ejecución para las aplicaciones sobre distintos tipos de hardware.
- Para ello proporcionan una serie de servicios a estas aplicaciones y a sus usuarios.
- Estos servicios varían de unos sistemas operativos a otros pero se pueden agrupar en grandes bloques.

1. Introducción

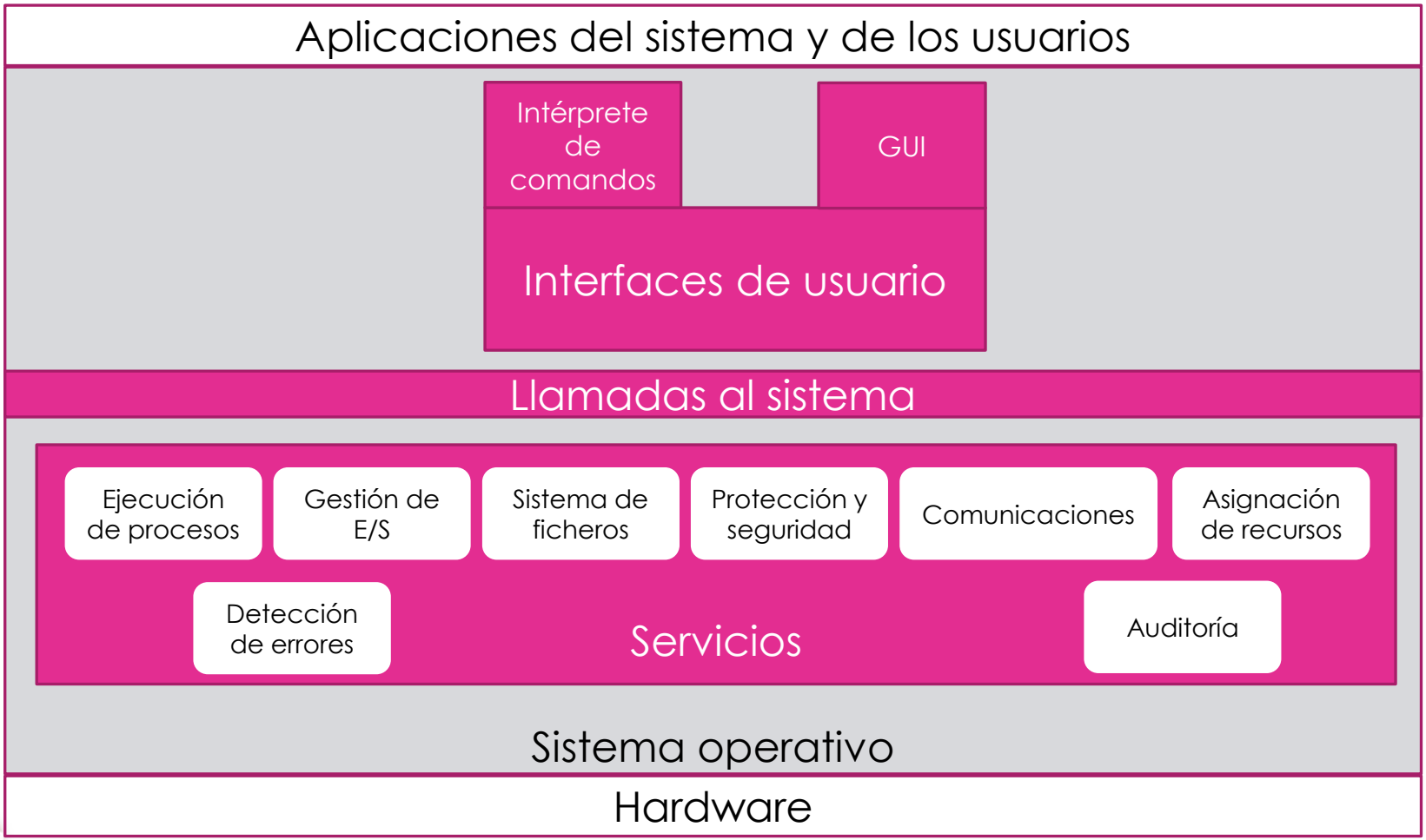
Hardware

Software

Comunicaciones

Datos/información

1. Introducción



2. Diseño de sistemas operativos y kernel

- Lo primero que debe tenerse en cuenta cuando se diseña un sistema operativo es el tipo de hardware para el que se diseña.
  - Esto puede imponer importantes restricciones al consumo de memoria o de energía (por ejemplo, en el caso de los dispositivos móviles).
  - O puede implicar que ciertos servicios, que no suelen ser muy comunes, se vuelvan imprescindibles (por ejemplo, en el caso de los sistemas distribuidos o empotrados).

2. Diseño de sistemas operativos y kernel

- A continuación deben tenerse en cuenta otras características de alto nivel:
  - Sistema operativo mono-usuario o multi-usuario.
  - Sistema operativo mono-proceso o multi-proceso.
  - Sistema operativo de propósito general o específico.
  - Sistema operativo de confianza (lo vemos en la próxima unidad).
  - Sistema operativo con requisitos de tiempo real.

2. Diseño de sistemas operativos y kernel

- A continuación se especifican los requisitos en dos grandes bloques.
  - Requisitos de usuario.
    - Asociados a los interfaces del SO (intérprete de comandos, GUI) y al rendimiento esperado, esencialmente.
  - Requisitos de sistema.
    - Que provienen de las consideraciones ya mencionadas a alto nivel pero también asociadas a la propia implementación del SO, a su mantenimiento y actualización, etc.

2. Diseño de sistemas operativos y kernel

- Con todo esto, no existe una metodología estándar para diseño de sistemas operativos.
- Aunque sí unos aspectos comunes que se han convertido en el estándar "de facto" en los últimos años.

Windows	UNIX/Linux	MacOS

2. Diseño de sistemas operativos y kernel

- Diseñar un sistema operativo, implica, al fin y al cabo (simplificando):
  - Identificar y diseñar el conjunto de mecanismos que permite que el SO cumpla los requisitos especificados (mediante el establecimiento de las políticas adecuadas) y proporcione los servicios necesarios.
  - Diseñar el conjunto de llamadas al sistema que se permite.
  - Diseñar las interfaces de usuario.

2. Diseño de sistemas operativos y kernel

- Una vez diseñado el sistema operativo, se pasa a la fase de implementación.
- Los primeros sistemas operativos tenían arquitecturas muy sencillas y se programaban directamente en ensamblador dada su cercanía al hardware del sistema.
- En la actualidad ya no se suele trabajar en ensamblador, sino que se usan lenguajes de programación de alto nivel como C o C++.
  - Aunque algunos fragmentos de código siguen programándose directamente en ensamblador.

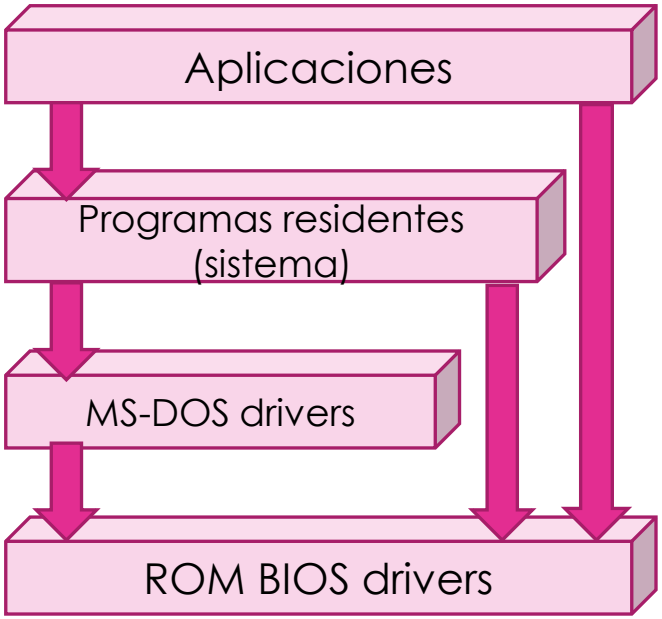
2. Diseño de sistemas operativos y kernel

- ¿Por qué se sigue recurriendo al ensamblador para algunas rutinas?
- Porque escribiendo directamente en ensamblador y ahorrándonos el paso de la compilación desde un lenguaje de alto nivel los códigos son más eficientes en el uso del hardware y se consiguen prestaciones más altas.
  - Así que se hace con las rutinas más críticas para el rendimiento del SO (en tiempo o en consumo de recursos).

Esto es muy interesante desde el punto de vista de la seguridad…

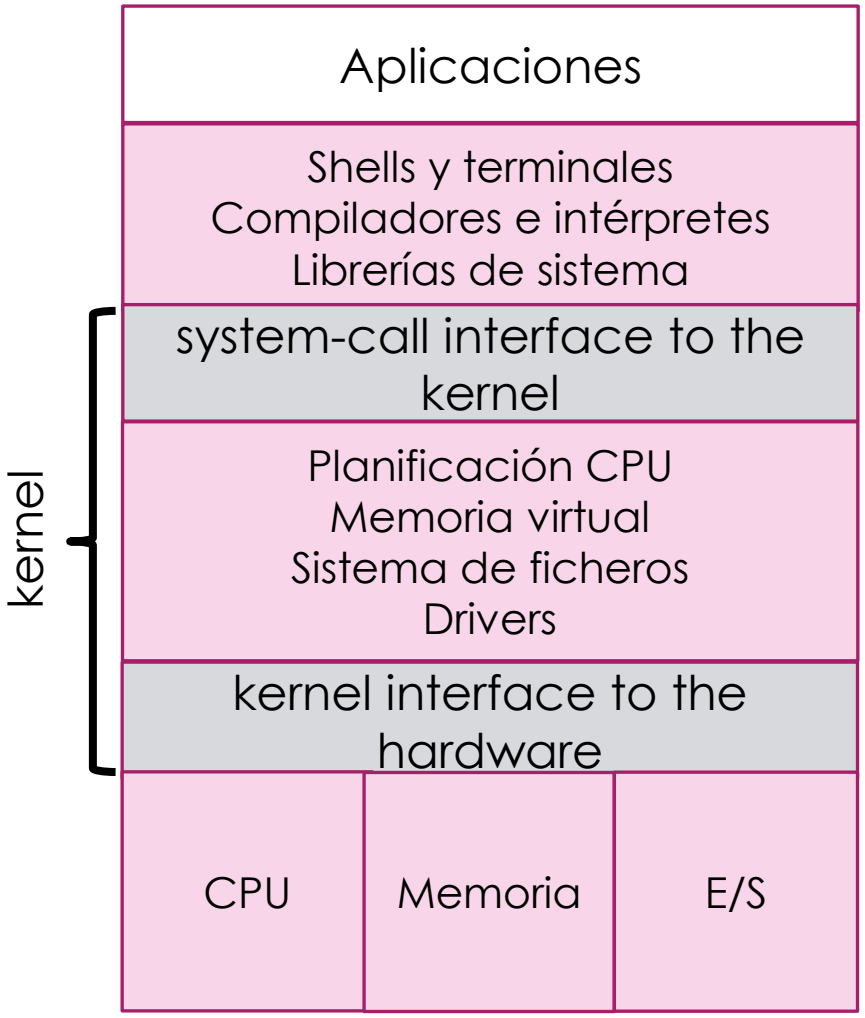
2. Diseño de sistemas operativos y kernel

- Si tomamos como ejemplo la implementación del sistema operativo MS-DOS, se puede ver que no se diseñó pensando en la importancia que iba a tener.
  - No es un sistema operativo especialmente flexible, ni fácil de modificar.
  - Los diferentes niveles de funcionalidad no están bien separados ni sus dependencias se han aislado de manera eficiente.
- Lo mismo pasa con las primeras implementaciones de UNIX.



MS-DOS

UNIX (primeras versiones)



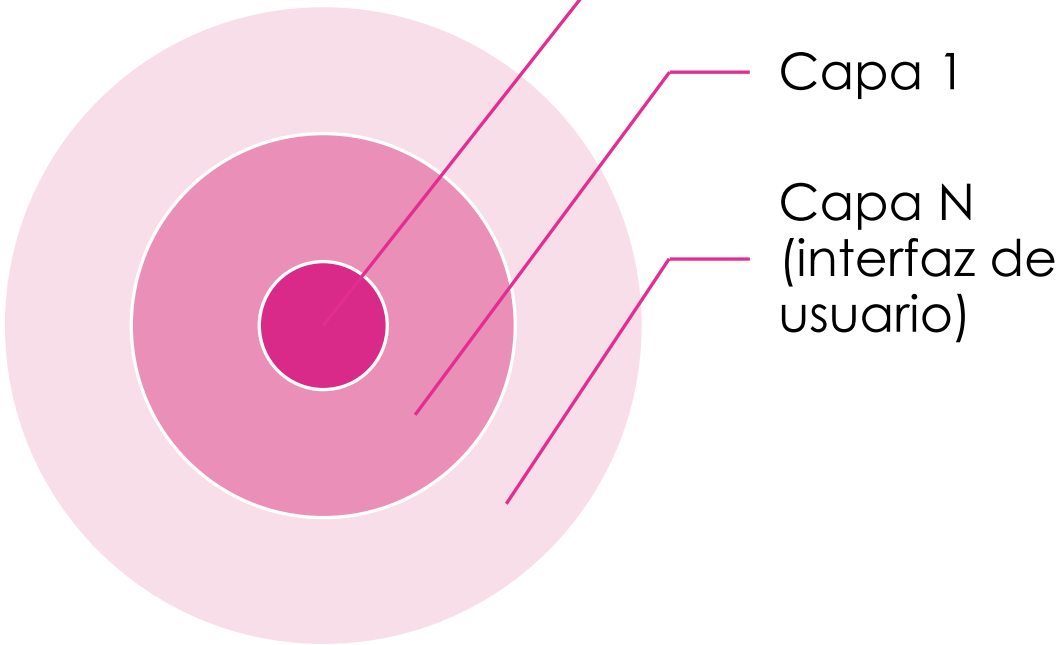
2. Diseño de sistemas operativos y kernel

- Este tipo de sistemas operativos suelen denominarse monolíticos.
  - Casi todas las funcionalidades están al mismo nivel, sin separación clara.
- En los sistemas operativos de tipo UNIX se comenzó a utilizar la palabra kernel para referirse a la implementación de los mecanismos esenciales para el SO.
  - Que en las primeras versiones era todo lo que estuviera entre el hardware y las llamadas al sistema.
  - El “modo kernel” de ejecución siempre es el prioritario (máximos privilegios).

2. Diseño de sistemas operativos y kernel

- Este tipo de sistemas monolíticos no fueron capaces de cumplir con las exigencias de las plataformas hardware que fueron surgiendo, por lo que se empezaron a proponer sistemas operativos modulares.
- En una primera aproximación, estos módulos se ordenaron por capas.
  - De más cerca del hardware a más cerca del usuario y de sus aplicaciones.

2. Diseño de sistemas operativos y kernel



Cada capa contiene estructuras de datos y rutinas que implementan los mecanismos del SO

2. Diseño de sistemas operativos y kernel

- En este tipo de implementación cada capa confía en las que están por debajo.
- No tiene que conocer los detalles de cómo están implementadas, simplemente, cómo invocar las rutinas que necesita y cómo se le va a responder.
- Esto simplifica la verificación y las modificaciones.
- Pero es muy difícil planificar la ubicación adecuada de las rutinas y mecanismos del SO en cada capa.

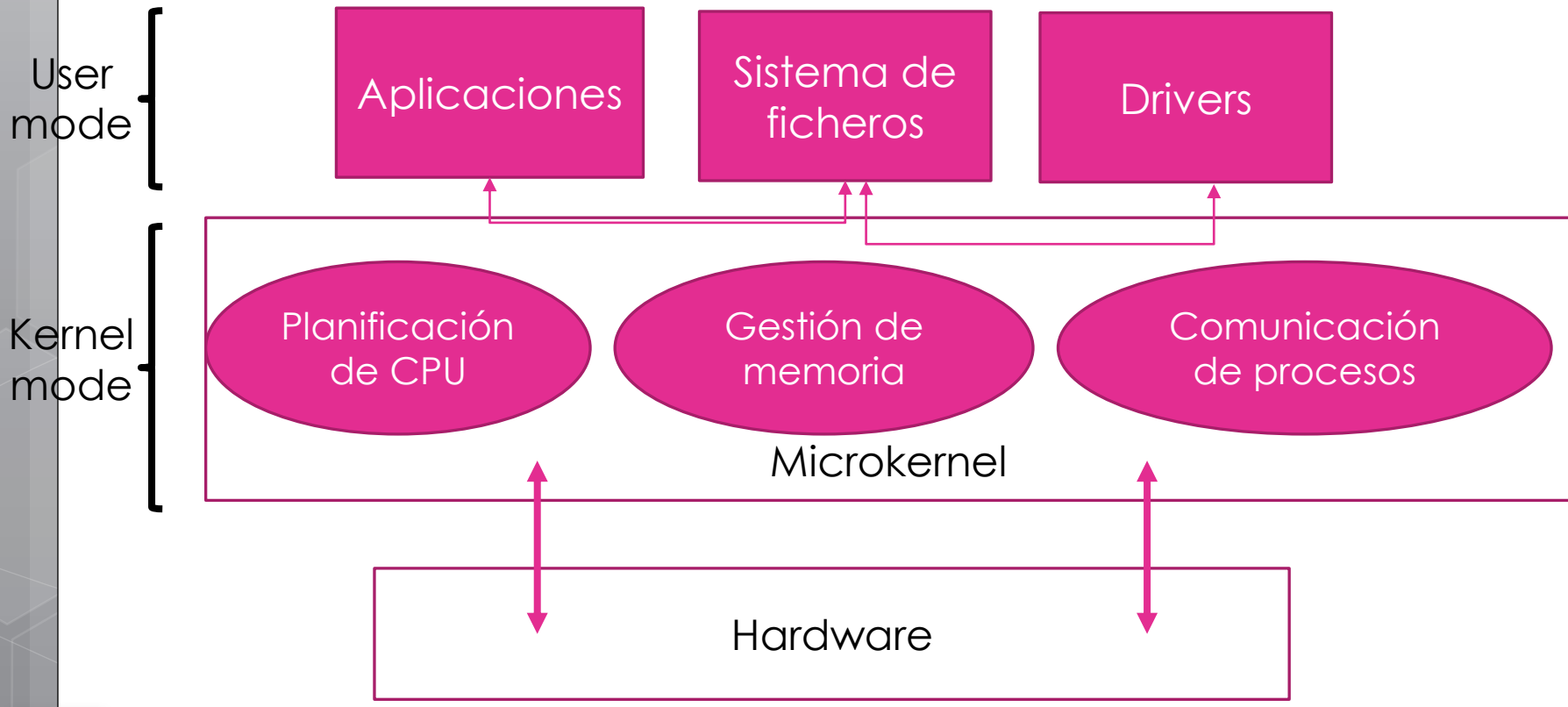
2. Diseño de sistemas operativos y kernel

- Otro problema de estos diseños por capas es el rendimiento.
  - Cada invocación de una capa superior a otra inferior conlleva una transformación de parámetros y una serie de llamadas.
  - Desde la capa más externa hasta la 0 esto puede implicar una latencia considerable.
- Por estos motivos se suele optar por pocas capas con bastante funcionalidad.
  - Tenemos el ejemplo de muchos sistemas de tipo UNIX.

2. Diseño de sistemas operativos y kernel

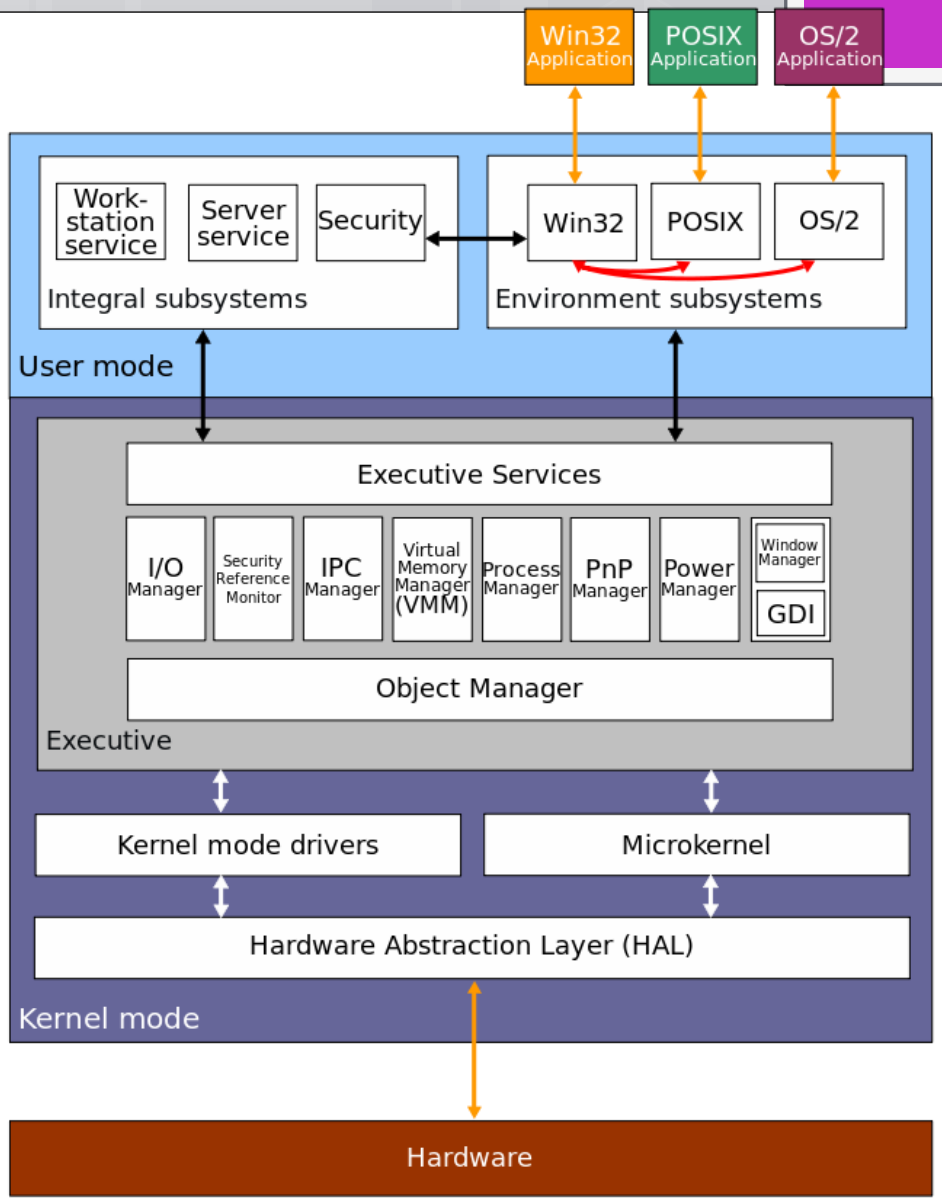
- Otra alternativa a los SO monolíticos son los basados en microkernel.
- En este caso se elimina del kernel del SO todo lo que no sea esencial.
  - Hasta quedarse con un microkernel.
- El resto de funcionalidades se implementan como procesos del sistema y programas de usuario, a un nivel superior.
  - SO fáciles de extender y de modificar y portables.

2. Diseño de sistemas operativos y kernel



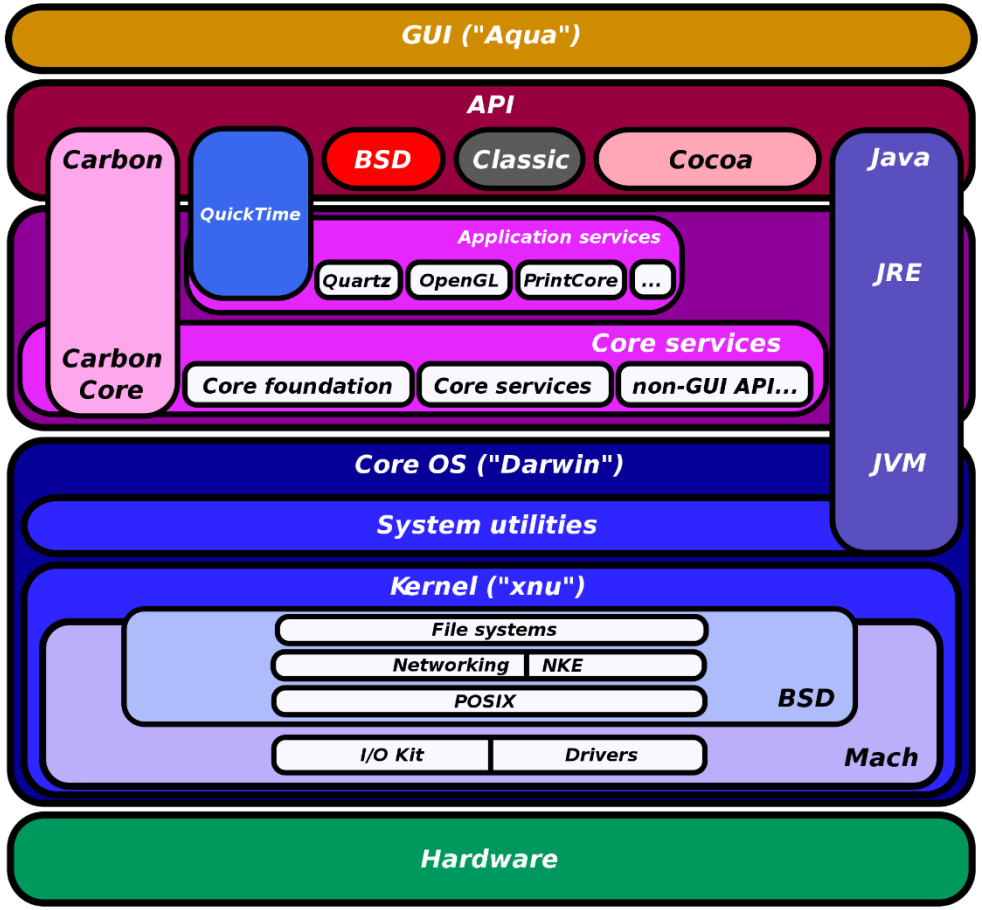
2. Diseño de sistemas operativos y kernel

- El microkernel Mach o el QNX (tiempo real) son los ejemplos más conocidos.
  - En el primero de ellos se basan todos los sistemas operativos de Apple.
- Windows NT también se implementó como un microkernel, pero sus problemas de rendimiento hicieron que se fueran incorporando funcionalidades que estaban programadas como procesos de sistema al microkernel y acabó siendo un SO monolítico.
  - Microsoft habla de kernel híbrido para todos sus SO.



Implementación de Windows NT y de los SO de Microsoft posteriores: kernel mode y user mode (ni monolítico del todo ni microkernel)

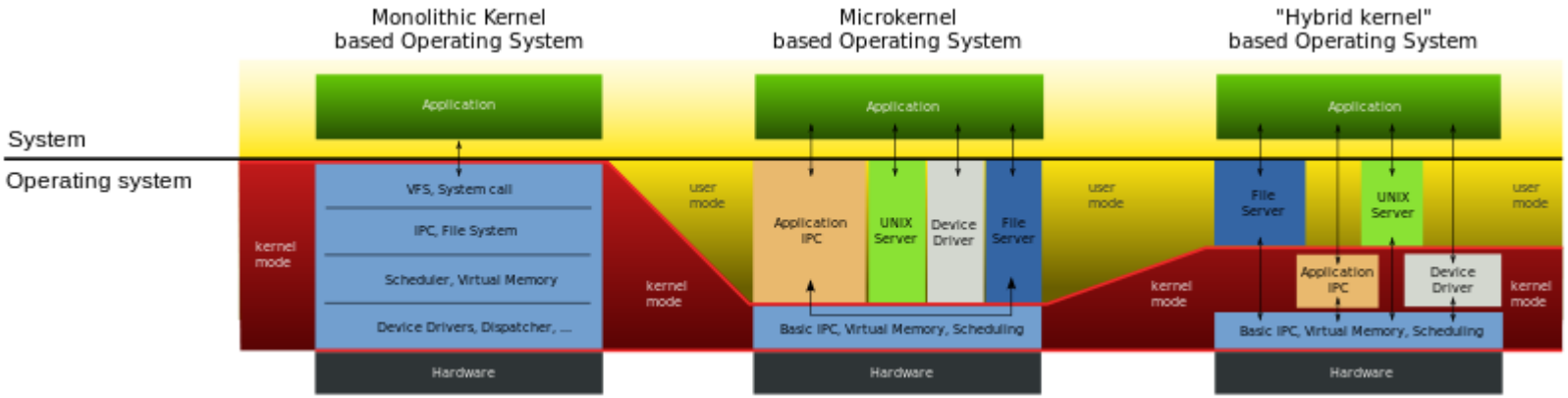
https://commons.wikimedia.org/wiki/File:Windows_2000_architecture.svg



Apple ha optado por lo mismo, añadiendo al microkernel Mach funcionalidades y componentes de BSD en un enfoque de kernel híbrido para XNU (iOS, tvOS y watchOS)

https://commons.wikimedia.org/wiki/File:Diagram_of_Mac_OS_X_architecture.svg

En resumen, en los sistemas operativos más extendidos en la actualidad, excepto en la mayor parte de distribuciones de Linux, tenemos un kernel híbrido:



https://commons.wikimedia.org/wiki/File:OS-structure2.svg

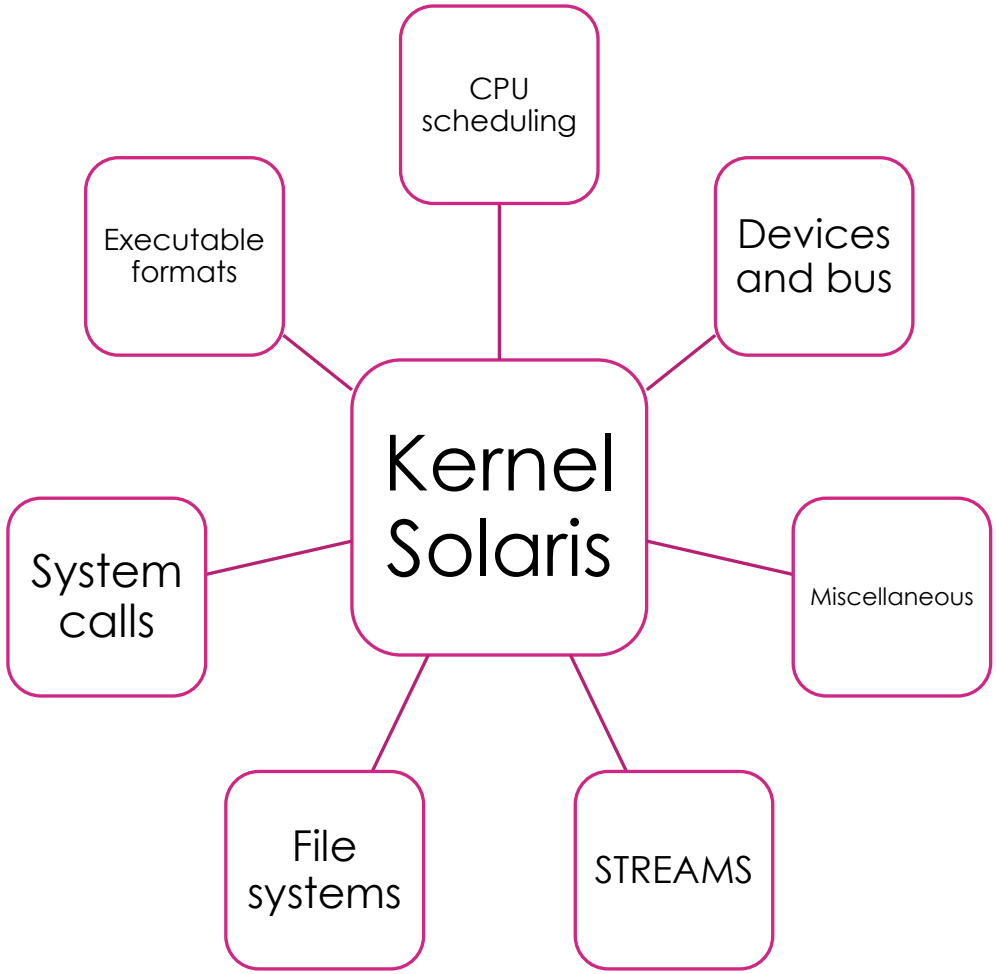
2. Diseño de sistemas operativos y kernel

- Por último, están los sistemas operativos modulares.
  - Casi todos las distribuciones Linux actuales y los sistemas operativos Solaris.
- Este tipo de sistemas operativos aprovechan las propiedades de los lenguajes de programación orientados a objetos.
- El kernel incorpora unos componentes o módulos básicos y durante el arranque del sistema o en tiempo de ejecución enlaza con otros módulos que extienden su funcionalidad.
  - De manera dinámica.

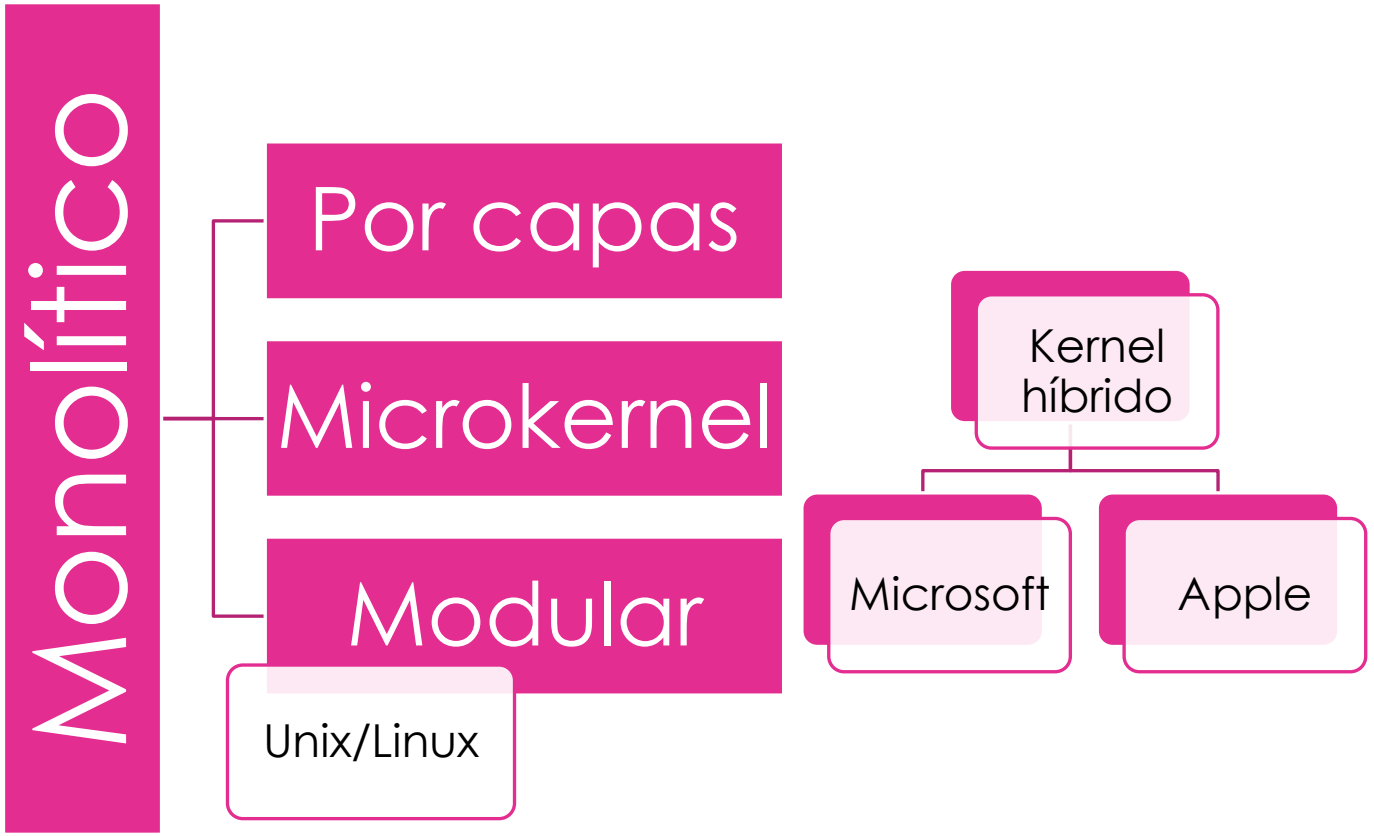
2. Diseño de sistemas operativos y kernel

- Este enfoque se parece mucho al del microkernel, confiando en un kernel básico que se puede extender cuando y como se necesite.
- Pero presenta un rendimiento mayor, ya que no necesita de paso de mensajes para comunicar estas “extensiones” con los mecanismos básicos que implementa el kernel.
- Los módulos suelen agruparse por funcionalidad.
- En esto se parece a los SO por capas, pero no hay ninguna obligación de respetar un orden o jerarquía en las invocaciones.

Por ejemplo, en Solaris hay siete tipos de módulos



2. Diseño de sistemas operativos y kernel



3. Llamadas al sistema

- Estas llamadas son las que permiten a los usuarios y aplicaciones invocar a los servicios ofrecidos por el sistema operativo.
- De nuevo pueden estar programadas en ensamblador (por motivos de rendimiento), en C o en C++.
- Los usuarios, administradores, desarrolladores, etc. pueden utilizar estas llamadas al sistema directamente.
  - Aunque lo habitual es utilizar comandos o APIs (Application Programming Interface) para abstraerles de los detalles a bajo nivel.

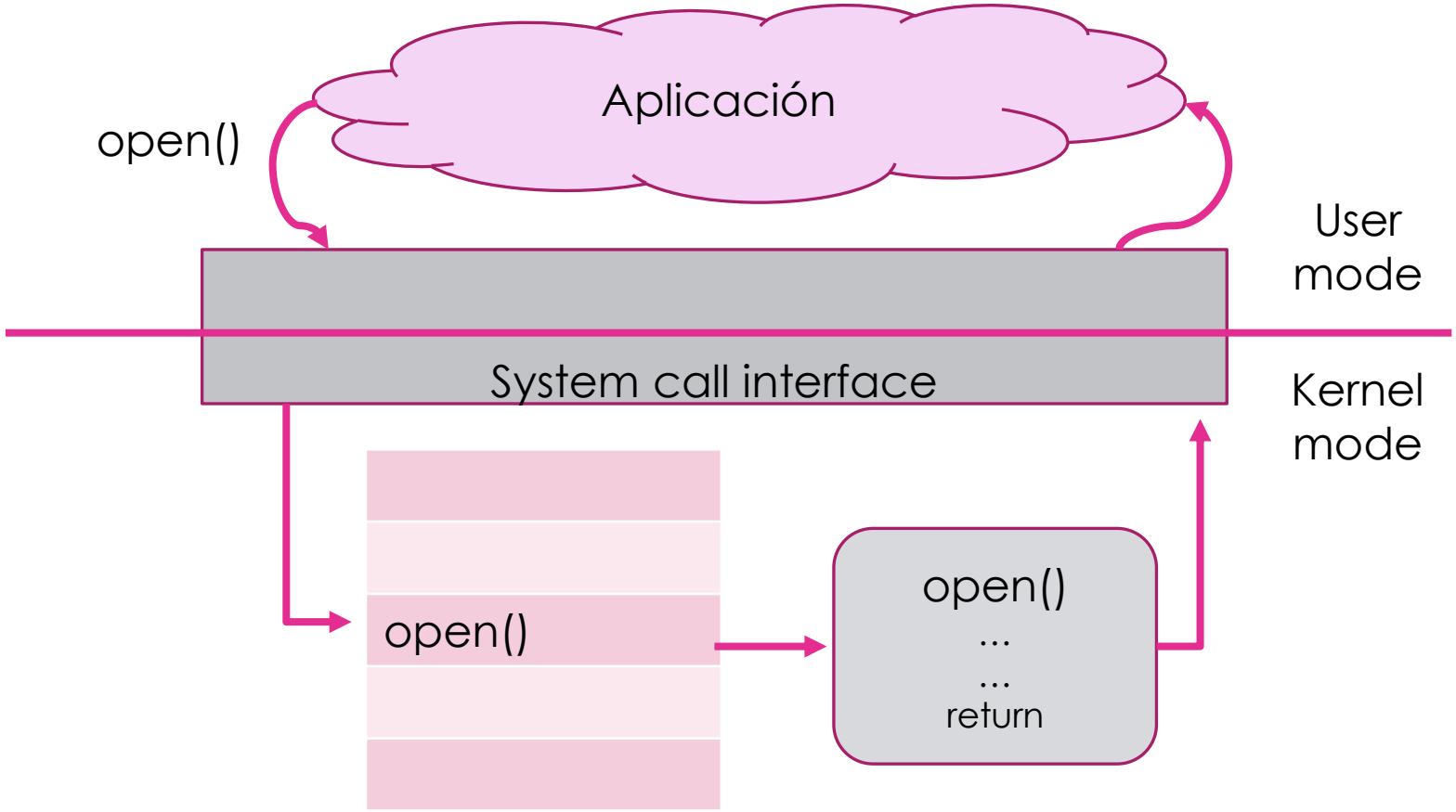
- Por ejemplo, el comando que permite copiar un fichero en realidad implica realizar todas estas llamadas al sistema:

Conseguir nombre del fichero origen
Conseguir nombre del fichero destino
Abrir el fichero origen, si no existe, abortar
Crear el fichero destino, si existe, abortar
Bucle: leer del fichero origen, escribir en el fichero destino, hasta que la lectura falle
Cerrar fichero destino
Terminar

3. Llamadas al sistema

- El uso de APIs hace que las aplicaciones presenten una mayor portabilidad de unos sistemas la otros, ya que las llamadas al sistema disponibles no son un conjunto estándar, cambian de unos sistemas operativos a otros.
  - Se suelen clasificar en seis grandes grupos.
  - El interfaz de llamadas al sistema se encarga de "traducir" las llamadas a la API a las llamadas al sistema disponibles.

3. Llamadas al sistema



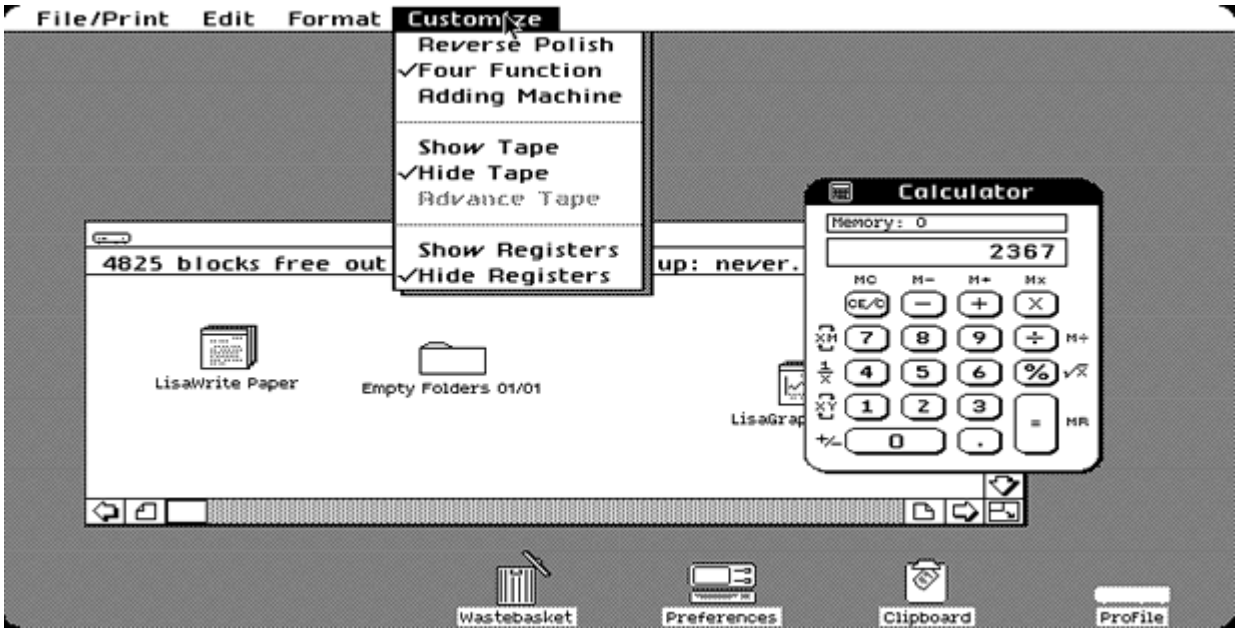
3. Llamadas al sistema

Control de procesos	Manipulación de ficheros	Gestión de dispositivos
Información	Comunicación	Protección

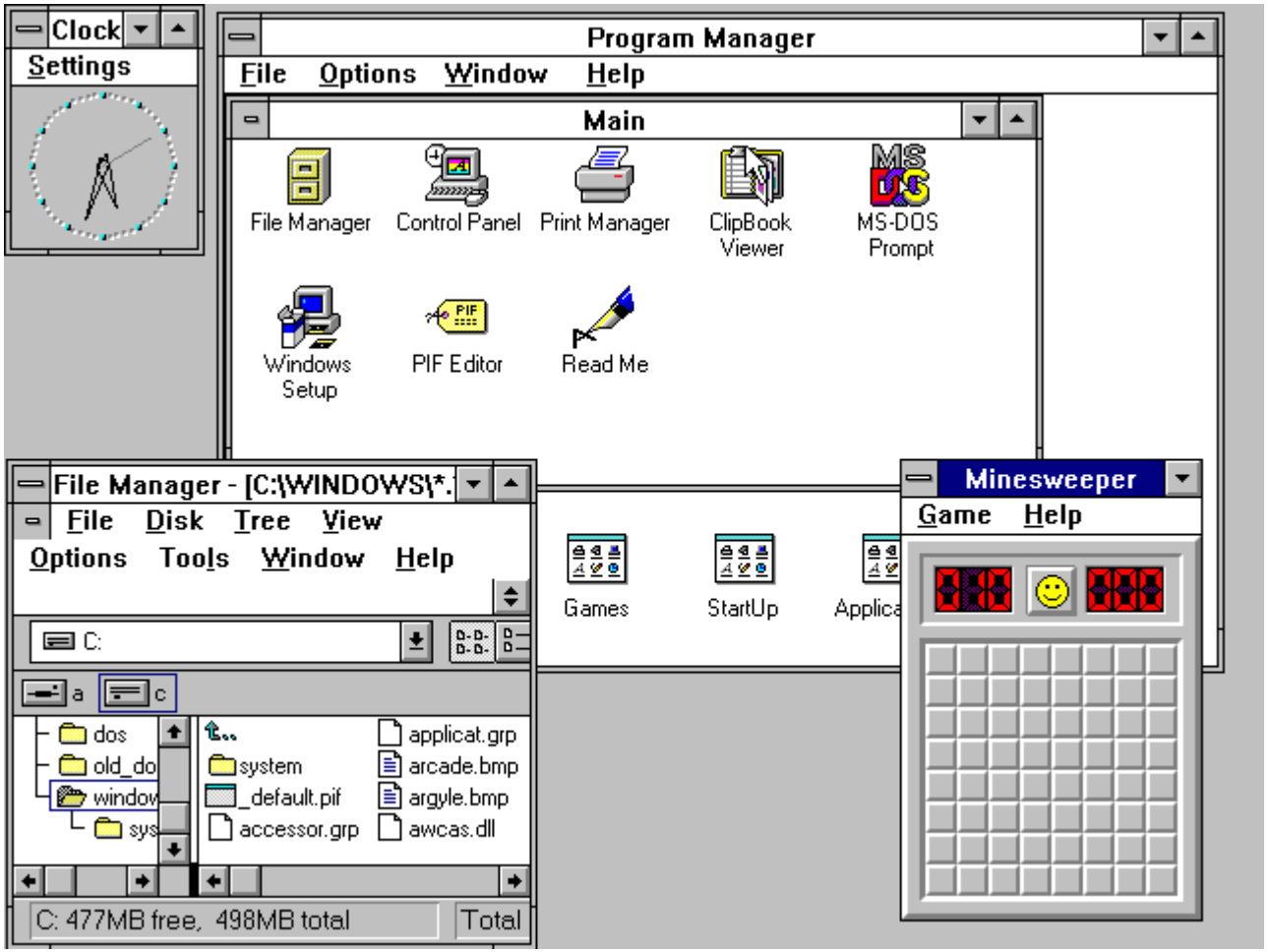
	Windows	Unix
Control de procesos	CreateProcess() ExitProcess()	fork() exit() wait()
Manipulación de ficheros	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Gestión de dispositivos	ReadConsole() WriteConsole()	ioctl() read() write()
Información	SetTimer() Sleep()	getpid() alarm() sleep()
Comunicación	CreatePipe() MapViewOfFile()	pipe() mmap()
Protección	SetFileSecurity() SetSecurityDescriptorGroup()	chmod() chown()

4. Interfaces de usuario

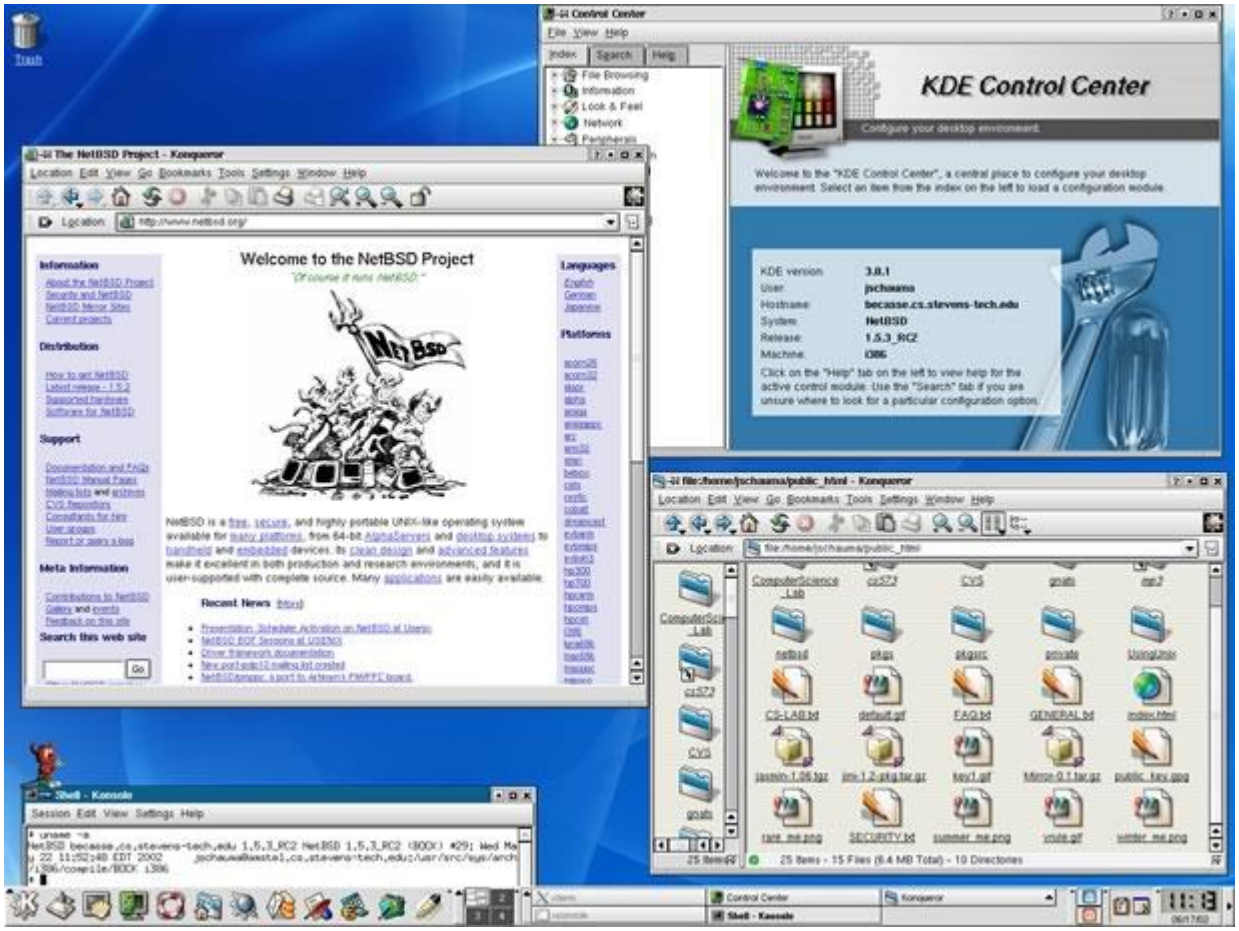
- Casi todos los sistemas operativos incorporan al menos dos:
  - Un intérprete de comandos.
    - Podría incluirse en el kernel pero suele ser una aplicación independiente.
    - Distintos tipos de consolas o shells permiten trabajar de manera diferente con los comandos y las llamadas al sistema.
  - Una interfaz gráfica o GUI.
    - Para interactuar mediante teclado/ratón o con pantallas táctiles mediante iconos, asistentes, etc.



GUI del Lisa OS de Apple (1983)



GUI de Windows3.1 de Microsoft (1992)



KDE 3 (2002)

Android 2.3.4 (2010)

4. El proceso de arranque

- Este proceso es esencial en todos los sistemas, y aunque depende de cada sistema operativo, hay unos pasos o fases que son bastante comunes.
- Cuando se enciende el hardware, cuando se conecta a la alimentación, se genera una interrupción que carga una determinada dirección de memoria en el contador de programa.
  - A partir de esta dirección, se carga en memoria la secuencia de arranque que se almacena en alguna de las memorias ROM de la placa base del sistema.

4. El proceso de arranque

- Esta secuencia suele identificar, primero, el hardware conectado y disponible tras ese arranque.
  - Y realiza los denominados POST o Power-On Self Tests.
- Durante esta fase inicial de chequeo, mediante alguna tecla especial, el usuario suele tener la posibilidad de acceder a la configuración de la BIOS o UEFI.
  - Configuraciones HW a bajo nivel.

4. El proceso de arranque

- A continuación la secuencia de arranque busca algún dispositivo de memoria no volátil en el que haya un sistema operativo instalado.
- Este dispositivo será por norma general un disco duro.
  - En el primer sector de este disco se puede consultar la tabla con las particiones de ese disco (donde empiezan, donde acaban, cómo están formateadas y desde cuál se arranca – partición activa-).
  - Además de esta tabla, suelen almacenarse en este primer sector las primeras instrucciones de la secuencia de arranque dependiente del SO.
    - Que re-dirigen a otra parte del disco para continuar con este arranque.

4. El proceso de arranque

- Esencialmente, se localiza el kernel del sistema operativo en el disco duro, se transfiere a la memoria del sistema y se le pasa el control.
  - Se puede dejar que el usuario escoja entre varios kernels, dejando uno por defecto.
  - También suelen permitirse opciones de arranque seguro, de recuperación o para tareas de mantenimiento.

4. El proceso de arranque

- El kernel inicia el proceso con PID 1, que inicializa las estructuras de datos del SO, carga el sistema de ficheros raíz, los drivers de los dispositivos de E/S, etc.
- También consulta el fichero de configuraciones para saber qué servicios del SO deben arrancarse.
- Por último, proporciona la opción de hacer login a los usuarios y se duerme, cediendo el control al planificador de la CPU.
- A partir de este momento, suele ejecutarse cuando se producen determinadas interrupciones.

5. Virtualización

- Las técnicas, mecanismos y tecnologías de virtualización permiten implementar recursos informáticos aislando unas capas del sistema de otras.
- De esta forma el alcance de cualquier cambio o modificación en una de las capas es mucho más restringido que en los sistemas tradicionales y casi no afecta a las demás.
- Además la virtualización puede implicar ahorro en otros tipos de costes.

5. Virtualización

Virtualización del hardware (o infraestructura ó máquina): La virtualización de máquina utiliza un software para crear una máquina virtual que emula los servicios y capacidades del hardware subyacente (procesamiento, memoria, almacenamiento, comunicaciones).

Virtualización de aplicaciones: En este caso se separa la aplicación del sistema operativo, lo que reduce los conflictos entre aplicaciones y simplifica las distribuciones y actualizaciones de software.

5. Virtualización

Virtualización de presentación: Permite que una aplicación en un equipo pueda ser controlada por otro en una ubicación diferente.

Virtualización de almacenamiento: En este caso los usuarios acceden a sus datos como si estuvieran en du disco duro local pero en realidad, es indiferente la localización de estos datos (discos duros virtuales).

Virtualización de red: Permite a los usuarios remotos navegar en la red de una empresa como si estuvieran conectados físicamente (red privada virtual).

5. Virtualización

- La virtualización de máquina permite la compartición de recursos hardware por diferentes usuarios y aplicaciones, mediante la multiplexación del tiempo de uso de dichos recursos.
- Cada instancia derivada de esta multiplexación es lo que se conoce como máquina virtual (VM).
- La idea es separar la capa hardware del resto de capas para:
  - Aumentar el rendimiento, la flexibilidad, la disponibilidad, la escalabilidad y la seguridad.
  - Favorecer la consolidación de los sistemas hardware y reducir costes.

5. Virtualización

- La virtualización a este nivel exige virtualizar, como mínimo:

Procesador	Memoria
Almacenamiento	Acceso a red

5. Virtualización

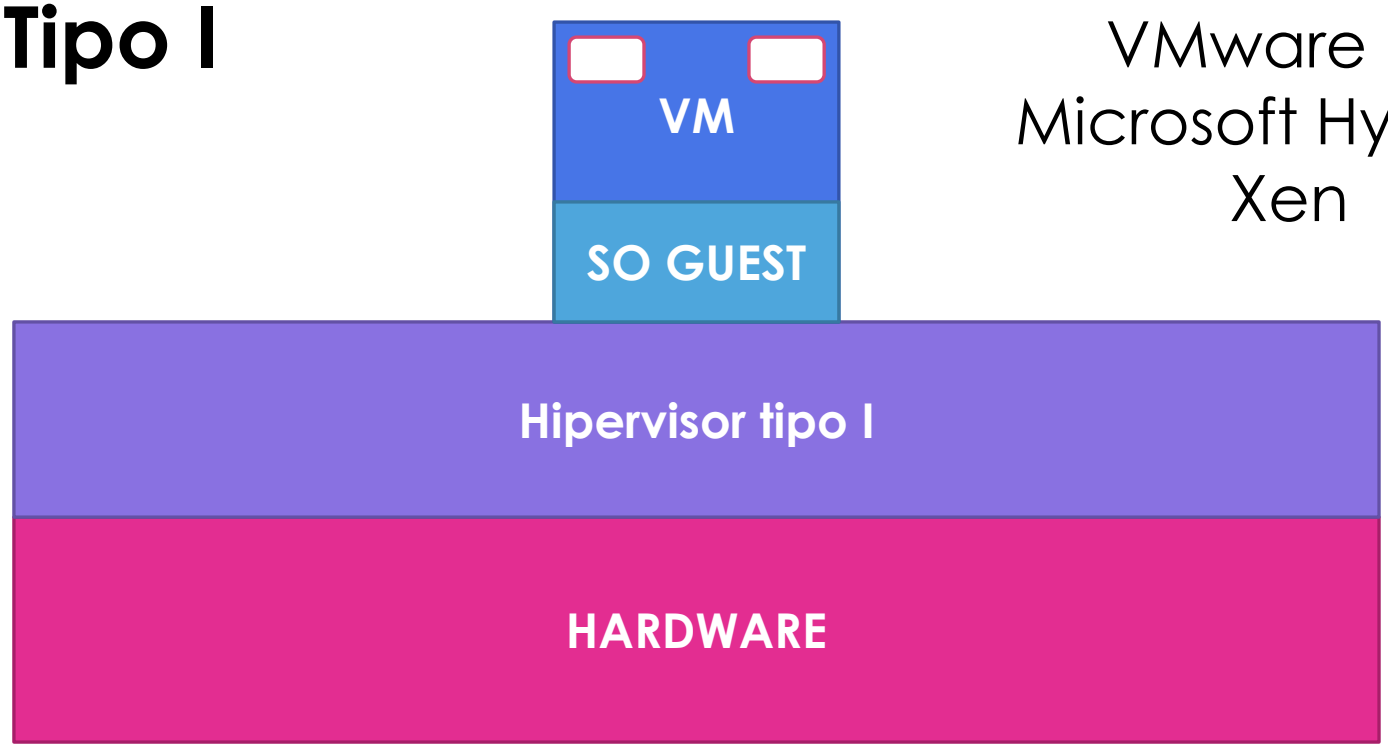
- Se denomina Virtual Machine Monitor (VMM) o hipervisor al software que proporciona el entorno en el que las máquinas virtuales operan.
- Debe garantizar como mínimo tres propiedades:

5. Virtualización

- Existe una clasificación inicial de hipervisores que atiende a su despliegue o instalación.
- Esta clasificación tiene que ver, sobre todo, con el número de capas que una aplicación que se ejecuta sobre una máquina virtual tiene que atravesar para llegar al hardware de la máquina real.
  - Hipervisores tipo I (o bare-metal).
  - Hipervisores tipo II.

5. Virtualización

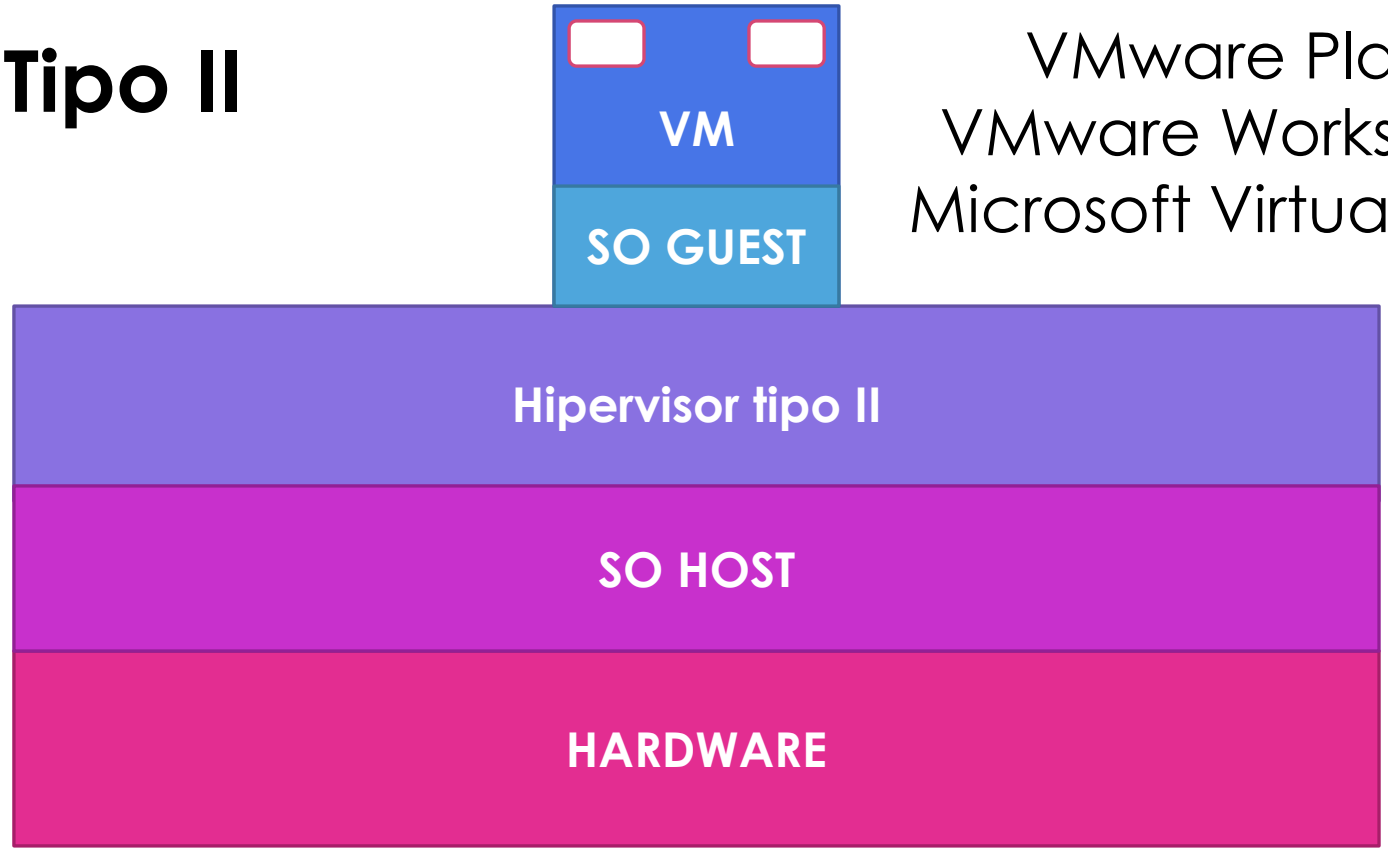
Tipo I



VMware ESX
Microsoft Hyper-V
Xen

5. Virtualización

Tipo II



VMware Player
VMware Workstation,
Microsoft Virtual Server

5. Virtualización

- En teoría la utilización de hipervisores de tipo I permite obtener configuraciones con un rendimiento mayor (al reducir las capas intermedias entre el sistema operativo guest y el hardware real) y de una mayor robustez (por no depender de un sistema operativo host).
- Además ahorran los costes del sistema operativo host y son más seguros.
  - Al evitar la propagación de errores y fallos por el SO host o incluso al reducir la superficie de exposición de las aplicaciones en las VM.

5. Virtualización

- Pero en mucho casos se opta por los hipervisores de tipo II dada su facilidad de instalación y configuración y sus garantías de acceso al hardware físico.
  - A través del sistema operativo host y de los drivers desarrollados para él.
- Esta clasificación de hipervisores se queda un poco corta hoy en día para categorizar la gran variedad de VMMs que han surgido.
  - Paravirtualización, contenedores, exokernels.

Para practicar un poco

1. ¿Qué sistema operativo utilizas habitualmente? Busca información sobre su kernel, cómo está diseñado, los mecanismos y servicios que ofrece, el conjunto de llamadas al sistema que proporciona, sus diferentes interfaces, etc.
2. En CVE, busca información sobre las últimas vulnerabilidades descubiertas para ese sistema operativo y verifica que has realizado las actualizaciones y parcheos oportunos.
3. ¿Te atreves a documentarte para que podamos discutir en clase qué sistema operativo nos parece "más seguro"?
4. Instala VirtualBox en tu PC (es un hipervisor gratuito de Oracle), utilizaremos este hipervisor a lo largo de todo el Grado. De momento puedes utilizarlo para probar diferentes sistemas operativos. Por ejemplo, puedes ir instalando una máquina virtual con Kali Linux.

Referencias

- Fotografías
  - https://unsplash.com
- Iconos
  - https://www.flaticon.es/

©2019-2022 Marta Beltrán URJC (marta.beltran@urjc.es)
Algunos derechos reservados.
Este documento se distribuye bajo la licencia “Reconocimiento-CompartirIgual 3.0 España” de Creative Commons, disponible en
https://creativecommons.org/licenses/by-sa/3.0/es/