Unidad 1: Conceptos generales de Sistemas Operativos.

Tema 3: Estructura del sistema operativo.

3.1 Componentes del sistema.
3.2 Servicios del sistema operativo.
3.3 Llamadas al sistema.
3.4 Programas del sistema.
3.5 Estructura del sistema.
3.6 Diseño e implementación de sistemas.
3.7 Generación de sistemas.

Estructura del Sistema Operativo:

- Fundamental:
  - Definir los objetivos del sistema antes de iniciar su diseño.
- Podemos considerar un S.O. desde varias perspectivas:
  - Servicios que proporciona.
  - Interfaz que ofrece a usuarios y programadores.
  - Componentes del sistema y sus interconexiones.

3.1 Componentes del sistema:

- **Sistema grande y complejo:**
  - División en componentes, con sus entradas, salidas y funciones cuidadosamente definidas:

3.1 Componentes del sistema:

- **3.1.1 Gestión de procesos:**
  - **Proceso:** programa o porción de programa en ejecución.
  - **Necesita** de ciertos **recursos**:
    - Tiempo de CPU.
    - Memoria.
    - Archivos.
    - Dispositivos de E/S.
    - Puede necesitar también datos de inicialización.
  - A la **finalización de un proceso** el SO **recupera los recursos** que le había asignado.
  - **Proceso:** Unidad de trabajo de un sistema. El stma consiste sólo en una colección de procesos (del SO o de usuario).

3.1 Componentes del sistema:

- **3.1.1 Gestión de procesos:**
  - El SO se encarga de las siguientes actividades relacionadas con la gestión de procesos:
    - Crear y eliminar procesos.
    - Bloquear y reanudar procesos.
    - Proveer mecanismos para la sincronización de procesos.
    - Proveer mecanismos para la comunicación de procesos.
    - Proveer mecanismos para manejar bloqueos mútuos.

3.1 Componentes del sistema:

- **3.1.2 Gestión de la memoria principal:**
  - **Memoria principal:** almacén de palabras o datos (cada uno con su propia dirección) de **acceso rápido**, que son **compartidos** por la CPU y los dispositivos de E/S.
  - Es el único dispositivo de almacenamiento grande que **la CPU puede direccionar y acceder directamente.**
  - Las **instrucciones** deben estar **en la MP** para que la CPU pueda ejecutarlas (es preciso cargar los programas en MP).
  - El SO se encarga de las siguientes actividades relacionadas con la gestión de memoria:
    - Saber qué partes de la memoria se están usando, cuáles están libres y quién las está usando (y el tamaño de cada parte).
    - Decidir **qué procesos cargar en la memoria**.
    - **Asignar y liberar espacio** de memoria.

- **3.1.3 Gestión de archivos:**
  - **Archivo:** conjunto de **información relacionada**, generalmente programas y datos. Se organizan en directorios para hacer su uso más sencillo. Cuando varios usuarios tienen acceso a los archivos, se debe controlar quién y de qué modo accede a ellos.
  - El SO se encarga de las siguientes actividades relacionadas con la gestión de archivos:
    - Crear y eliminar archivos.
    - Crear y eliminar directorios.
    - Proveer las primitivas para manejo de archivos y directorios.
    - Establecer la correspondencia archivo-almacenamiento secundario.
    - Guardar los archivos en almacenamientos no volátiles.

- **3.1.4 Gestión del sistema E/S:**
  - Sistema de E/S: Conjunto de dispositivos muy variados y complejos de programar.
  - El SO se encarga de las siguientes actividades relacionadas con la gestión del sistema E/S:
    - Proporcionar una interfaz uniforme para el acceso a los dispositivos.
    - Proporcionar manejadores para los dispositivos concretos.
    - Tratar automáticamente los errores más típicos.
    - Para los dispositivos de almacenamiento, usar cachés.
    - Para los discos, planificar de forma óptima las peticiones.

3.1.5 Gestión de almacenamiento secundario:

- **Problemas de la MP:** pequeña (no tienen cabida todos los datos y programas) y volátil.
- **Almacenamiento secundario no volátil**, casi todos los programas (compiladores, ensambladores, rutinas de ordenación, editores, ...) se guardan en disco hasta que se cargan en memoria.
- El SO se encarga de las siguientes actividades relacionadas con la gestión de discos:
  - Administración del espacio libre.
  - Asignación del almacenamiento.
  - Planificación del disco.

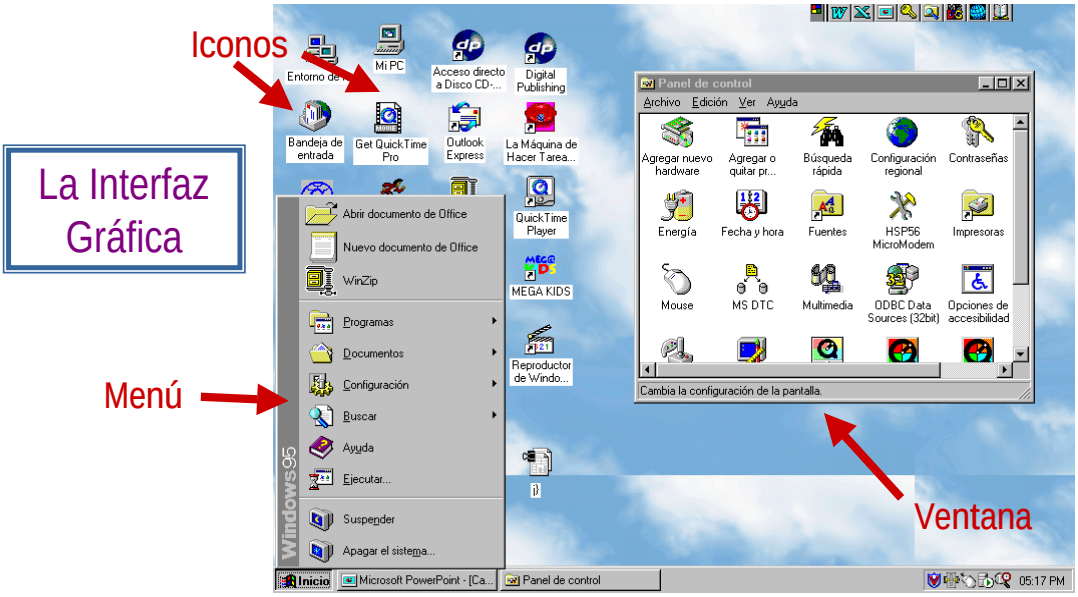
3.1.6 Trabajo con redes:

- **Sistema distribuido:** colección de procesadores con sus propios recursos locales (memoria local, reloj) y que se comunica con otros procesadores conectados mediante una red.
- Objetivos del SO:
  - Proporcionar primitivas (de comunicación) para **conectarse con equipos remotos** y acceder de forma controlada a sus recursos:
    - Primitivas de comunicación (envío y recepción de datos).
    - Sistema de ficheros en red (NFS).
    - Llamada remota a procedimiento (RFC), etc.

- **3.1.7 Sistema de protección:**
  - Es preciso proteger a un proceso de los demás, ya que el sistema de computación admite **múltiples usuarios** y la **ejecución concurrente** de procesos.
  - **Protección:** Mecanismo para **controlar el acceso** de programas, procesos o usuarios a los recursos definidos por un sistema de computador.
  - La protección puede **mejorar la confiabilidad** mediante la detección de errores.

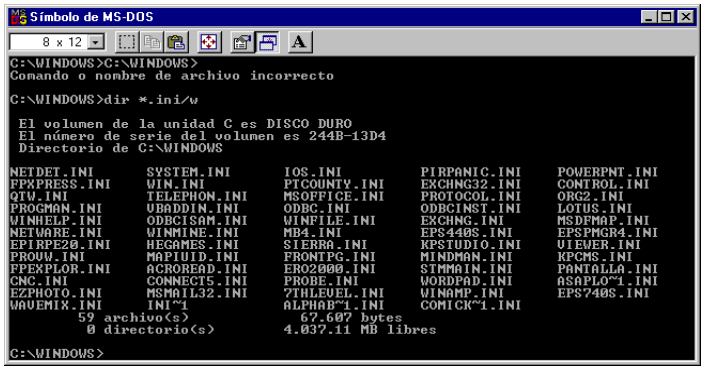
- **3.1.8 Sistema de interpretación de órdenes:**
  - **Interfaz entre usuario y sistema operativo**. Para que un usuario pueda **dialogar** directamente con el SO.
  - Se proporciona una interfaz de usuario básica para:
    - Cargar programas y abortar programas.
    - Introducir datos a los programas.
    - Trabajar con archivos y trabajar con redes.
  - Ej: JCL stmas por lotes, COMMAND.COM MS-DOS, shell de UNIX.
  - Puede ser de dos tipos:
    - Gráfica (GUI: Graphic User Interface).
    - De línea de comandos.
  - La interfaz gráfica:
    - Su uso se basa en:
      - Metáfora de un escritorio, donde se muestran objetos gráficos para representar los recursos disponibles.
      - El ratón como dispositivo de entrada.
      - Un grupo de herramientas gráficas: Ventanas (representan programas y sus archivos asociados), iconos (representan los recursos del ordenador) y menús (listas de comandos relacionados entre sí).

3.1 Componentes del sistema:



3.1 Componentes del sistema:

- **3.1.8 Interpretación de órdenes: Interfaz de línea de comandos:**
  - Su uso se basa en:
    - El conocimiento, por parte del usuario, de los comandos que pertenecen al Sistema Operativo
    - El teclado como dispositivo de entrada.
    - La línea latente en la pantalla, donde el usuario debe escribir cada comando (prompt).



3.2 Servicios del sistema operativo:

- **Servicios a los programas y a sus usuarios:**
  - Ejecución de programas.
  - Operaciones de E/S.
  - Manipulación del sistema de archivos.
  - Comunicaciones: entre procesos y de red (técnicas de memoria compartida y paso de mensajes).
  - Detección de errores.

- **Asegura el funcionamiento eficiente del sistema:**
  - Asignación de recursos: varios usuarios – varios trabajos ejecutándose a la vez.
  - Contabilización: qué usuarios usan qué recursos.
  - Protección: controlar accesos a los recursos del sistema.
  - Seguridad: cada usuario debe identificarse ante el sistema.

3.3 Llamadas al sistema:

- Interfaces con los servicios del sistema operativo:
  - Para el programador: llamadas al sistema en lenguaje máquina o en alto nivel.
  - Para el usuario:
    - Intérprete de órdenes.
    - Programas del sistema.
- El SO ofrece una gama de servicios a los programas, que acceden a ellos mediante llamadas al sistema.
- Son la interfaz entre el programa en ejecución y el SO.
- Única forma en la que un programa puede solicitar operaciones al SO.
- Ejemplo de llamadas al sistema:
  - UNIX: fd = open (“mifichero”, O_RDONLY);

3.3 Llamadas al sistema:

- Implementación de las llamadas al sistema:
  - ¿Cómo se implementa la llamada?
    - Habitualmente, mediante una instrucción especial de la máquina (syscall, int, trap, ...).
    - La instrucción cambia automáticamente a modo privilegiado.
    - Si programamos en un lenguaje de alto nivel escribimos la llamada al sistema como una subrutina, y el compilador la sustituye por la instrucción de máquina correspondiente.
  - Muchas llamadas necesitan parámetros ¿cómo los pasamos al SO?:
    - Usando registros de la máquina.
    - En una tabla en memoria principal.
    - Poniéndolos en la pila (stack):

3.3 Llamadas al sistema:

- **Tipos de llamadas al sistema:**
  - **Control de procesos:**
    - Fin, abortar, cargar, ejecutar, crear, finalizar, obtener y establecer atributos, espera, asignar y liberar memoria.
  - **Manipulación de archivos:**
    - Crear y eliminar archivo, abrir y cerrar, leer, escribir, reposicionar, obtener y establecer atributos.
  - **Manipulación de dispositivos:**
    - Solicitar y liberar, leer, escribir, reposicionar, obtener y establecer atributos, conectar y desconectar dispositivos.
  - **Mantenimiento de información:**
    - Obtener y establecer hora, fecha, datos del sistema, atributos de un proceso, archivo o dispositivo.
  - **Comunicaciones:**
    - Crear-eliminar conexiones; enviar-recibir mensajes, transferir información de estado,conectar-desconectar dispositivos remotos

3.4 Programas del sistema:

- Las llamadas al sistema proporcionan una interfaz para el programador. El usuario final interactúa con el SO mediante programas previamente compilados.

- El entorno del SO provee de utilidades básicas para:
  - Manipular ficheros.
  - Editar documentos.
  - Proporcionar un entorno de trabajo.
  - Desarrollar programas (compiladores, enlazadores, etc.).
  - Comunicarnos con otros equipos (telnet, ftp, ssh, etc).

- Núcleo (kernel) del SO:
  - Software que **reside permanentemente en memoria** y que atiende las llamadas al sistema y demás eventos básicos.
  - Distinción entre núcleo y programas del sistema (que son los que utilizan los servicios del núcleo).

3.5 Estructura del sistema:

- El **núcleo** (***kernel***) es el componente central de la mayoría de los sistemas operativos.
  - Entre las responsabilidades del núcleo se encuentran
    - la gestión de los recursos del sistema, y,
    - la comunicación entre los componentes hardware y software
  - Como componente básico de un sistema operativo, el núcleo proporciona el nivel más bajo en la jerarquía de capas de abstracción en relación con los recursos (especialmente, memoria, procesadores y dispositivos de E/S) que las aplicaciones deben controlar, para realizar sus funciones.
  - Típicamente, el núcleo ofrece sus servicios:
    - Bien, a través de los mecanismos de comunicación entre procesos,
    - Bien, a través de la interfaz de llamadas al sistema
- Estas tareas se hacen de forma diferente, dependiendo de su diseño e implementación. Básicamente, las dos filosofías existentes son:
  - Los **núcleos monolíticos** (***monolithic kernels***) tratan de alcanzar estos objetivos mediante la ejecución de todo el código en el mismo espacio de direcciones, con el fin de incrementar el rendimiento del sistema.
  - Los **micronúcleos** (***microkernels***) ejecutan la mayoría de sus servicios en el espacio de usuario, con la intención de mejorar la mantenibilidad y modularidad del código del sistema.

3.5 Estructura del sistema:

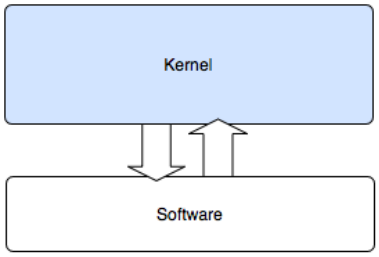
- La mayoría de los sistemas operativos confían en el concepto de núcleo
  - La existencia de un núcleo es consecuencia natural del sistema operativo como una **jerarquía de capas de abstracción**, donde la implementación de cada capa descansa sobre la funcionalidad de las capas inferiores
  - Desde este punto de vista, el núcleo simplemente es el nombre que se da a la capa de abstracción más baja implementada como software
- En la mayoría de los casos, el cargador del sistema operativo ejecuta el núcleo en modo supervisor
  - El núcleo, entonces, se inicializa a sí mismo y crea el primer proceso
  - Después de esto, el núcleo no se ejecuta directamente, sino sólo en respuesta a eventos externos (vía las llamadas al sistema utilizadas por las aplicaciones para requerir servicios del núcleo, vía interrupciones utilizadas por el hardware para notificar al núcleo la ocurrencia de determinados eventos)
  - Además, el núcleo implementa un proceso "vacío" (*idle process*) consistente en un bucle infinito, que se ejecuta cuando no hay ningún otro proceso disponible para su ejecución
- El desarrollo del núcleo se considera como una de las tareas más complejas y difíciles en programación
  - Su posición tan relevante dentro del sistema operativo, obliga a que el núcleo, necesariamente, tenga un buen rendimiento, y esto hace que sea, por tanto, una pieza de software crítica en lo que se refiere a su correcto diseño e implementación

3.5 Estructura del sistema:

- Aparte de gestionar eficaz y eficientemente los recursos del computador, el núcleo también proporciona mecanismos para la sincronización y **comunicación entre procesos** (**IPC**, ***inter-process communication***)
- El núcleo puede implementar todas estas funcionalidades el mismo o confiar en algunos procesos que el núcleo ejecuta, para proporcionar estas facilidades al resto del procesos
  - En este último caso, el núcleo debe proporcionar algunos mecanismos IPC para permitir al resto de procesos acceder a las facilidades proporcionadas por cada uno de estos procesos "ayudantes"
- Para que un proceso de usuario realice realmente un trabajo útil, debe ser capaz de acceder a los servicios proporcionados por el núcleo. Esto se implementa de forma diferente por parte de cada núcleo, pero la mayoría proporciona una biblioteca de funciones en C o una API (*Application Programming Interface*), conocida genéricamente como **interfaz de llamadas al sistema**, que a su vez invocan a las funciones del núcleo correspondientes

3.5 Estructura del sistema:

- **Núcleos monolíticos** (***monolithic kernel***)
  - Un núcleo monolítico se basa en una arquitectura en la que la totalidad del núcleo se ejecuta en el espacio de memoria propio del núcleo, en modo supervisor
  - Al igual que otras arquitecturas (micro-núcleos o núcleos híbridos), el núcleo define una interfaz de alto nivel sobre el hardware de la computadora, con un conjunto de primitivas o llamadas al sistema para acceder a los servicios del sistema operativo, tales como gestión de procesos, concurrencia, y gestión de memoria, en uno o más módulos



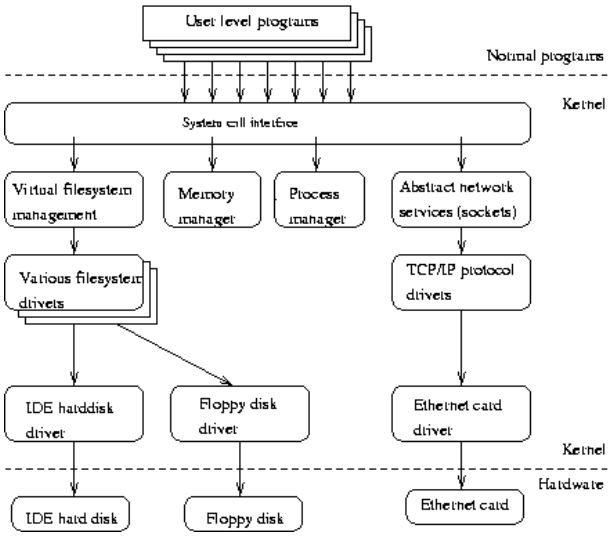
  - Incluso cuando cada módulo que da servicio a estas operaciones, está separado del resto, resulta muy difícil alcanzar la total integración del código, y, dado que todos los módulos se ejecutan en el mismo espacio de direcciones, un error en un módulo, puede afectar a todo el sistema
  - Sin embargo, cuando la implementación es completa y fiable, la estrecha integración de sus componentes, se traduce en una implementación muy eficaz y eficiente

3.5 Estructura del sistema:

- **Núcleos monolíticos** (***continuación***)
  - La mayoría de los núcleos monolíticos modernos (Linux, FreeBSD y Solaris), pueden cargar (descargar) dinámicamente módulos ejecutables en tiempo de ejecución
    - Esta modularidad del núcleo es a nivel de código ejecutable y no lo es a nivel arquitectónico del núcleo (este último nivel de modularidad es característico de los micro-núcleos o núcleos híbridos)
    - En la práctica, la carga de módulos de forma dinámica es simplemente una forma más flexible de manejar la imagen del núcleo en tiempo de ejecución, en contraposición, a re-arrancar el SO con una imagen del núcleo diferente
    - A pesar de que esta característica introduce una pequeña sobrecarga (*overhead*) respecto a que la funcionalidad del módulo estuviera internamente implementada, la carga de los módulos sólo cuando se necesiten, puede ayudar a mantener la cantidad de código que se ejecuta en el espacio del núcleo en el mínimo posible
  - Ejemplos de sistemas basados en núcleos monolíticos son:
    - Núcleos de sistemas Unix tradicionales: BSDs y Solaris; y Linux
    - MS-DOS, sistemas Microsoft Windows 9x (Windows 95, Windows 98 y Windows 98SE)
    - Mac OS, (hasta la versión Mac OS 8.6)

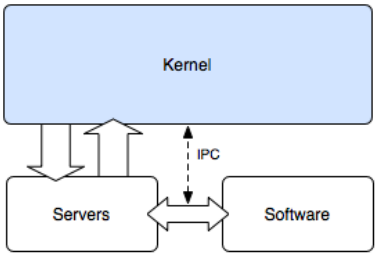
3.5 Estructura del sistema:

- **Núcleos monolíticos** (***continuación***)
  - Ejemplo: estructura núcleo Linux



3.5 Estructura del sistema:

- **Micro-núcleos** (***Microkernels***)
  - Un micro-núcleo es un núcleo de SO mínimo que proporciona sólo servicios básicos del SO (llamadas al sistema), mientras que otros servicios (habitualmente proporcionados por el núcleo de los SO convencionales) son proporcionados por programas de usuario, denominados ***servidores***



  - Comúnmente, los micro-núcleos proporcionan servicios tales como la gestión del espacio de direcciones, gestión de hilos y comunicación entre procesos, pero no los servicios relacionados con la conectividad de redes o la visualización, por ejemplo
  - Las ventajas de la aproximación basada en micro-núcleos al diseño del sistema son:
    - La inclusión de un nuevo servicio no requiere la modificación del núcleo
    - Es tanto más seguro, cuantas más operaciones se realicen en modo usuario en vez de en modo supervisor
    - Generalmente, un diseño del núcleo más sencillo y funcional, se traduce en un sistema operativo más fiable
  - Ejemplos de SO basados en micro-núcleos: AmigaOS, Amoeba, Minix

3.5 Estructura del sistema:

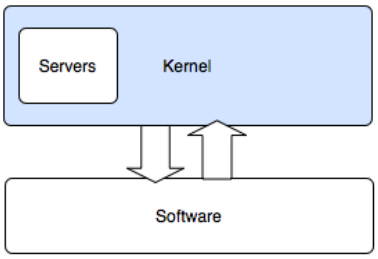
- **Micro-núcleos** (***continuación***)
  - **Comunicación entre procesos (IPC)**
    - La comunicación entre procesos (IPC) se refiere a cualquier mecanismo que permite a procesos diferentes, que se ejecutan sobre el mismo sistema operativo, intercomunicándose mediante el envío de mensajes
    - Esta característica permite que el sistema operativo se construya sobre un conjunto pequeño de programas, denominados servidores, que son utilizados por otros programas del sistema
    - El sistema de comunicación IPC es un componente clave de la arquitectura micro-núcleo, incluso más que en los núcleos monolíticos
  - **Servidores**
    - Los servidores en una arquitectura basada en micro-núcleo son programas como otros cualquiera, salvo que el núcleo les concede algunos privilegios para interactuar con partes de memoria que, de otro modo, estarían fuera de los límites de la mayoría de los programas
      - Esta característica permite a algunos servidores interactuar directamente con el hardware
    - Un conjunto básico de servidores para un micro-núcleo de propósito general incluyen servidores de sistemas de archivo, servidores de *drivers* de dispositivos, servidores de interconexión de redes, servidores de visualización y servidores de dispositivos de interfaz de usuario
    - Además, muchos fallos del sistema pueden corregirse, simplemente, parando y re-arrancando el servidor
      - En un sistema tradicional, un fallo del sistema en cualquier parte del código residente del núcleo puede causar un fallo general de la máquina, forzando el re-arranque de la máquina

3.5 Estructura del sistema:

- **Núcleos monolíticos** vs. **Micro-núcleos**
  - Los micro-núcleos, generalmente, tienen un menor rendimiento que los diseños tradicionales
    - Esto se debe en gran parte, a la sobrecarga derivada de mover a través del núcleo, todos los datos que se comunican las aplicaciones de usuario y los servidores, lo cual implica un cambio de contexto
  - Un micro-núcleo permite la implementación del resto del sistema operativo, como si fuera un programa de aplicación “normal”, escrito en un lenguaje de alto nivel, y permite el uso de diferentes sistemas operativos sobre un único núcleo
  - Los núcleos monolíticos se caracterizan porque el espacio de memoria requerido por el núcleo, se incrementa conforme crecen las funcionalidades implementadas por el núcleo
  - Los núcleos monolíticos tienden a ser más fáciles de diseñar, si bien, los errores en un sistema monolítico implican, habitualmente, el fallo del sistema completo

3.5 Estructura del sistema:

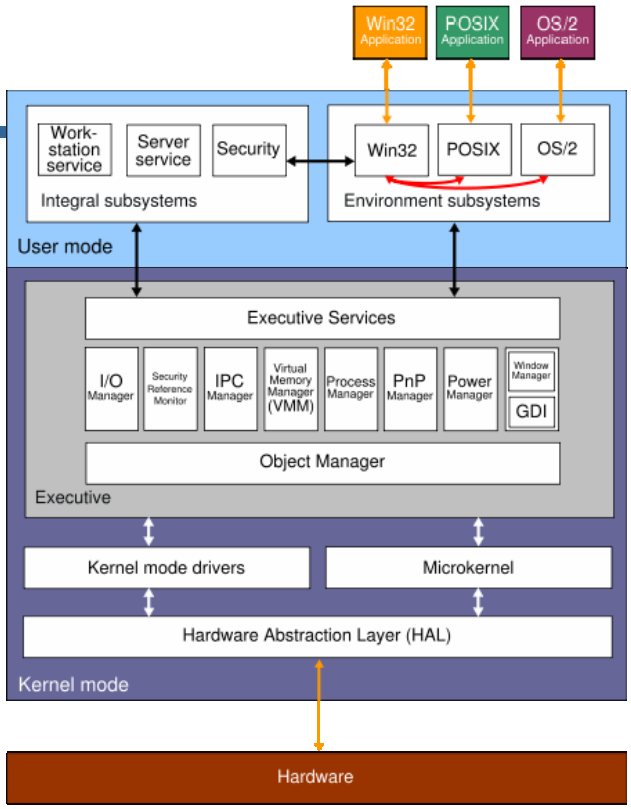
- **Núcleos híbridos**
  - Los núcleos híbridos constituyen una arquitectura de diseño, basada en la combinación de aspectos de las arquitecturas basadas en núcleos monolíticos y micro-núcleos
  - Se trata de una arquitectura controvertida debido a la similitud con los núcleos monolíticos (ciertos autores defienden que son básicamente lo mismo, y que su motivación es puramente originada por una cuestión de *marketing*)



  - La idea detrás de esta arquitectura es disponer de una estructura del núcleo similar a un micro-núcleo, pero implementada como un núcleo monolítico
    - A diferencia de un micro-núcleo, todos o casi todos los servicios están en el espacio del núcleo
    - Como en el caso de los núcleos monolíticos, no hay sobrecarga en el rendimiento derivado del paso de mensajes y cambio de contexto entre modo usuario y núcleo, propio de los micro-núcleos

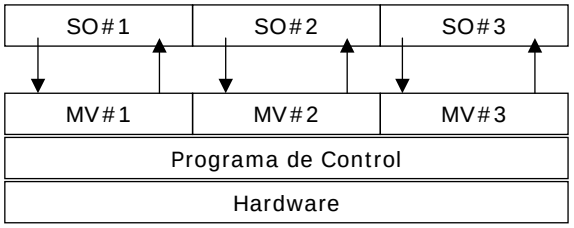
3.5 Estructura del sistema:

- **Núcleos híbridos (*continuación*)**
  - El ejemplo más conocido de núcleo híbrido es el núcleo NT, incluido en los sistemas Windows NT, W2K, XP y 2003. Todos los servicios se ejecutan en el espacio del núcleo



3.5 Estructura del Sistema:

- **Máquinas Virtuales**
  - **Descripción**: crea ilusiones (máquinas virtuales) de la máquina real, permitiendo que en cada máquina virtual se ejecute un S.O. distinto
    - El programa de control es el que se ejecuta directamente sobre el propio hardware y ofrece al nivel inmediatamente superior, varias máquinas virtuales
    - El software emulador convierte las peticiones hechas a la máquina virtual en operaciones sobre la máquina real.
    - Se pueden ejecutar varias máquinas virtuales al mismo tiempo (ej. mediante tiempo compartido).
    - Los recursos reales se reparten entre las máquinas virtuales.

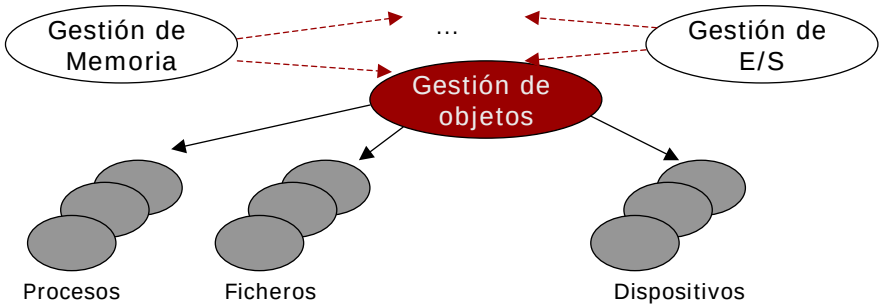


3.5 Estructura del Sistema:

- **Máquinas virtuales (continuación)**
  - **Ejemplos**:
    - IBM VM: ofrecía a cada usuario su propia máquina virtual monotarea; las máquinas virtuales se planifican con tiempo compartido.
    - JVM: Máquina virtual Java en la que se ejecutan los programas compilados en Java.
    - VMWare: en un PC, es capaz de ejecutar al mismo tiempo varias sesiones Windows, Linux, OS/2, etc.
    - Nachos: SO que se ejecuta en una máquina virtual MIPS, cuyo emulador corre sobre UNIX.
  - **Ventajas e inconvenientes de las máquinas virtuales:**
    - Protección: cada máquina virtual está aislada de las otras y no puede interferir.
    - Investigación y desarrollo: se puede desarrollar y ejecutar para un hw que no tenemos.
    - Independencia del hardware (ej: Java).
    - Pervivencia de sistemas antiguos (ej: emuladores, MS-DOS).
    - La implementación de la memoria virtual puede ser compleja y lenta.

3.5 Estructura del Sistema:

- **Estructura orientada a objetos**
  - **Descripción**: se basan en una colección de objetos, donde las funciones del sistema son un tipo de objeto (ficheros, dispositivos, etc.). La interacción entre dichos objetos viene determinada por las capacidades que cada uno tenga para actuar con el otro.
    - El kernel es el responsable del mantenimiento de las definiciones de los tipos de objetos soportados y del control de los privilegios de acceso a los mismos. Cuando un programa desee realizar una operación sobre un objeto determinado, deberá ejecutar una llamada al sistema, indicando qué derechos tiene para poder utilizarlo y qué operación intenta llevar a cabo. Como resultado de dicha llamada, el sistema validará la petición y, si puede ser aceptada, permitirá la realización de dicha operación.



3.6 Diseño e implementación de sistemas:

- Objetivos del diseño:
  - **Definición de objetivos y especificaciones:**
    - **Selección del hardware.**
    - **Tipo de procesamiento**: por lotes, de tiempo compartido, mono/multiusuario, distribuido, de tiempo real, ...
    - Mayor complicación tiene **especificar los requisitos** a cumplir:
      - Metas del usuario:
        - Cómodo y fácil de usar y de aprender, confiable, seguro, rápido.
      - Metas del sistema (diseñar, crear, mantener y operar):
        - Diseño, implementación y mantenimiento fácel, flexible, confiable, libre de errores, eficiente.
  - Gran **complicación** para definir los **requisitos del SO**, que serán diferentes según cada tipo de sistema.
  - **Ingeniería del software**: se ocupa de especificación y diseño.

3.6 Diseño e implementación de sistemas:

- Mecanismos y políticas:
  - Los **mecanismos** determinar **cómo se hace algo**.
  - Las **políticas** deciden **qué se hace**.
  - Ejemplo: un mecanismo para asegurar la protección de la CPU es la construcción de un temporizador; la decisión de a qué intervalo de tiempo se ajustará el temporizador para un usuario en particular es una decisión de política.
  - Es **conveniente separar los mecanismos de las políticas**. Aunque las políticas cambien, es deseable que los mismos mecanismos pueden seguir siendo útiles.

3.6 Diseño e implementación de sistemas:

- Implementación de sistemas:
  - Tradicionalmente los SO se han escrito en lenguaje ensamblador (por eficiencia).
  - Actualidad: uso de lenguajes de alto nivel; ej: UNIX, OS/2 y Windows NT están escritos principalmente en C (también se usa bastante C++).
    - Ventajas:
      - Más legible y fácil de mantener y depurar.
      - Más transportable a distintas arquitecturas hardware.
    - Desventajas:
      - Menor velocidad.
      - Mayor necesidad de almacenamiento.
  - Nota: la programación de un SO tiene la complicación de las pruebas. Para salvarle se usan emuladores.

3.7 Generación de sistemas:

- Hay que determinar los siguientes tipos de información:
  - ¿Qué CPU se usará?.
  - ¿Qué opciones están instaladas?.
  - ¿Qué memoria se tiene?.
  - ¿Con qué dispositivos se cuenta?.
  - ¿Opciones y parámetros a usar?.
- Tras determinar esta información se incluye en el código fuente y se compila por completo el SO. Tras generar el SO, debe ponerse a disposición del hw. ¿Cómo sabe el hw dónde está el núcleo o cómo debe cargarlo?.
- El computador se inicia cargando un núcleo, “arranque del sistema”. El programa de arranque (guardado en ROM) localiza el núcleo, lo carga en memoria e inicia la ejecución.