Gato_Kernel
www.wuolah.com/student/Gato_Kernel



# Tema 1.pdf

*Resumen Temario*

**2º Sistemas Operativos**

**Grado en Ingeniería Informática**

**Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**
**UGR - Universidad de Granada**

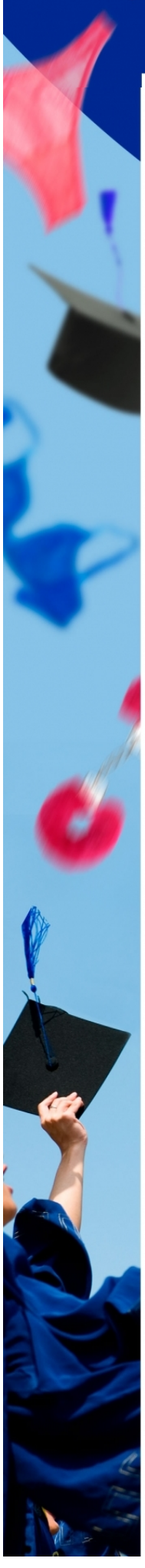
# Resumen tema 1

## Tipos de Sistemas Operativos

- **Por lotes:** un monitor, almacenado en memoria, se encarga de cargar un trabajo también almacenado en memoria, ejecutarlo y cargar el siguiente. Las tarjetas de control son el archivo de entrada que indican lo que hay que hacer para ejecutar el trabajo. Su problema principal es la larga espera entre lotes de trabajos.
- **Multiprogramados:** se benefician de la asincronía de los dispositivos de E/S, mantienen varios trabajos en memoria y solapan procesamiento de E/S de un proceso con cómputo de otro. Sin embargo, necesitan interrupciones o DMA.
- **De tiempo compartido:** soportan el uso interactivo del sistema, por lo que cada usuario tiene la ilusión de disponer de la máquina completa.
- **De tiempo real:** garantizan una respuesta a sucesos físicos en intervalos de tiempo fijos.
- **Sistema distribuido:** sistema multicomputador que no tiene memoria ni reloj común. Comparten recursos distribuidos, como hardware (impresoras) o software (web). Permiten un aumento de la fiabilidad del sistema, ya que si una parte falla el resto pueden seguir ejecutándose.
- **Sistemas paralelos:** sistemas de computador con varias CPUs que comparten una única memoria y reloj. Sustentan aplicaciones paralelas que intentan obtener un aumento de velocidad en tareas computacionales complejas. Ejemplo: simulación de explosión atómica. Todos los procesadores pueden ejecutar código del SO y de las aplicaciones.
- **Sistemas de Internet:** el computador de red suministra una funcionalidad mínima para ejecutar un navegador de Internet que carga los *applets* necesarios para dotar de la funcionalidad necesaria.

  Un **proceso** es la instancia de un programa en ejecución. El Sistema Operativo es el responsable de:
  - Crear y destruir procesos, así como suspenderlos y reanudarlos o suministrarle mecanismos para la comunicación entre ellos.
  - Asignar/desasignar memoria a los programas, mantener una pista sobre el uso de memoria y gestionarla (proporcionar y retirar).

  Un **archivo** es una colección de información con nombre. Un sistema de archivos suministra las primitivas para manipular archivos, backup, cuotas, etc.
  Un **manejador de dispositivo** es un módulo del Sistema Operativo que gestiona un tipo de dispositivo. Encapsula el conocimiento específico sobre él (inicialización, interrupciones, etc).
  La **protección** hace referencia al mecanismo para controlar los accesos de los programas a los recursos del sistema. Suele ser un mecanismo general a todo el SO, no está localizada en un único módulo.
  Un **intérprete de órdenes** es un programa o proceso que maneja la interpretación de órdenes del usuario desde un terminal o script para acceder a los servicios del SO.
  Un buen Sistema Operativo debe ser;
- **Eficiente**, ya que consume muchos ciclos de CPU.
- **Fiable**
  - **Robusto**, debe responder de forma predecible a condiciones de error, incluidos fallos hardware.
  - Debe protegerse activamente a sí mismo y a los usuarios de acciones malintencionadas o accidentales.
- **Extensible :** su funcionalidad debe poder ser modificada o extendida de forma sencilla.



Estructuras de Sistemas Operativos

- **Monolíticos:** no tienen una estructura clara y bien definida, sino que todos sus componentes se encuentran integrados en un único programa ( el Sistema Operativo ) que se ejecuta en un único espacio de direcciones. Además, todas las funciones se ejecutan en modo kernel. Ejemplos: MS-DOS y UNIX. El problema de esta estructura es la gran dificultad de modificación para añadir nuevas funcionalidades y servicios y que no se oculta la información.
- **Estructurados:** hay dos tipos:
  - **Sistemas por capas:** el Sistema Operativo se organiza mediante una jerarquía de capas, donde cada una ofrece una interfaz clara y bien definida a la capa superior y únicamente emplea los servicios que le ofrece la capa inferior. Ventajas: modularidad y ocultación de la información, ya que una capa no necesita conocer la implementación de las inferiores, sino únicamente su interfaz. Ésto facilita la depuración y verificación. El primer sistema en usar esta estructura fue THE, formado por 6 capas.
  - **Cliente-servidor:** la mayor parte de los servicios y funciones del sistema se implementan con el objetivo de ser ejecutados en modo usuario, dejando sólo una pequeña parte del Sistema Operativo ejecutable en modo kernel. Esta parte se llama *micronúcleo*, mientras que los procesos que ejecutan el resto de funciones se denominan *servidores*.
    No hay una definición clara de las funciones que debe llevar a cabo el micronúcleo. La mayoría incluyen la gestión de interrupciones, procesos y memoria y servicios básicos de comunicación entre procesos. Para solicitar un servicio en este tipo de sistema, el proceso de usuario (cliente) tiene que solicitárselo al servidor del sistema correspondiente. A su vez, el servidor puede requerir servicios de otros servidores, convirtiéndose en cliente.
    Este modelo presenta una gran flexibilidad, ya que cada proceso únicamente realiza una funcionalidad concreta, lo que hace que cada parte sea pequeña y manejable. Ésto facilita el desarrollo y la depuración de los servidores. Por otro lado, la fiabilidad aumenta, ya que un fallo únicamente afecta a un módulo.
    Como desventaja principal, el sistema se ve más sobrecargado ya que cada proceso utiliza un espacio de direcciones distinto, por lo que la activación requiere más tiempo. Ejemplos: Mac OS X, Windows NT.
    Ésta estructura tiene peor rendimiento que la monolítica, ya que para obtener un servicio se realizan muchos cambios de modo y espacio de direcciones.

- **Máquina virtual:** consiste en un monitor capaz de suministrar un número de versiones del hardware, es decir, máquinas virtuales. Cada copia es una réplica exacta del hardware, incluso con modo kernel y usuario, por lo que sobre cada una de ellas se puede instalar un Sistema Operativo. Cada petición de servicio es atendida por la copia del SO sobre la que se ejecuta. Cuando ese SO quiere acceder al hardware, se comunica con la máquina virtual como si se tratase de una real. Sin embargo, realmente se comunica con el monitor de máquina virtual, el único con acceso a la máquina real. Otra forma de construir una máquina virtual es aquel en que el monitor se ejecuta sobre un Sistema Operativo en lugar de sobre el hardware.

  Ventajas:
  - Añade multiprogramación, ya que cada máquina ejecuta sus procesos.
  - Añade mayor aislamiento, ya que si se compromete la seguridad en un SO no se compromete en el resto.
  - Si se genera una máquina estándar, no hay que recompilar para pasar de una máquina a otra.

Inconveniente : añade una sobrecarga computacional, por lo que la ejecución puede ser más lenta. Son difíciles de implementar (dos modos del procesador también en MV).
En algunos diseños se añade una capa ( *exokernel )* que se ejecuta en modo privilegiado y se encarga de asignar recursos a las máquinas virtuales y garantizar que ninguna utilice los recursos de otra.
Dada la importancia de este asunto, algunos procesadores incluyen mecanismos hardware para facilitar la construcción de máquinas virtuales ( *Intel Virtualization Technology ).*

- **Distribuido :** es un sistema diseñado para gestionar un multicomputador. El usuario percibe un único SO centralizado, haciendo por tanto más fácil el uso de la máquina. Un SO Distribuido tiene las mismas características que uno convencional pero aplicadas a un multicomputador. Estos sistemas no tuvieron éxito y se quedaron en fase experimental.
  - *Middleware:* es una capa de software que se ejecuta sobre un SO existente y que gestiona un sistema distribuido. Presenta una funcionalidad similar a la de un SO distribuido, pero se ejecuta sobre SO ya existentes que pueden ser distintos. Ejemplo: DCOM.

El **espacio de direcciones** de un proceso son aquellas direcciones de memoria que el proceso puede direccionar.
El Sistema Operativo confina la ejecución de un proceso a su espacio de direcciones.
El Sistema Operativo es parte del espacio de direcciones de todos los procesos.
Una llamada a un procedimiento es posible si su dirección está en el espacio de direcciones del llamador.

La comunicación entre los procesos se consigue mediante el **paso de mensajes**:
1. La aplicación envía el mensaje de solicitud (*send*) y espera la respuesta (*receive).*
2. El kernel verifica el mensaje y lo entrega al servidor.
3. El servidor que estaba en espera realiza el servicio y devuelve el resultado.
4.

La mayoría de los núcleos de Linux son monolíticos, es decir, incluyen prácticamente toda la funcionalidad del Sistema Operativo en un gran bloque de código que se ejecuta como un único proceso con un sólo espacio de direccionamiento. Si se hace algún cambio sobre alguna porción del Sistema Operativo, todos los módulos y rutinas tendrán que volverse a enlazar e instalar, además de tener que reiniciar el sistema. Esto conlleva una dificultad añadida al añadir por ejemplo algún controlador de dispositivo o función del sistema.
Sin embargo, Linux emplea una arquitectura modular particular: está estructurado como una colección de bloques que pueden cargarse y descargarse bajo demanda. Estos bloques se denominan módulos cargables. Un módulo es un fichero cuyo código puede enlazarse y desenlazarse con el núcleo en tiempo de ejecución. Normalmente implementa algunas funciones específicas (controlador de dispositivo, características de la capa superior del núcleo, sistema de archivos, etc.). Un módulo no se ejecuta como un nuevo proceso, sino en modo núcleo en nombre del proceso que lo carga, aunque puede crear hebras para llevar a cabo su propósito.
Los módulos cargables tienen dos características importantes:

- **Enlace dinámico:** un módulo puede cargarse, ejecutarse y descargarse mientras el núcleo está en memoria y ejecutándose.
- **Módulos apilables:** los módulos se gestionan como una jerarquía. Actúan como bibliotecas cuando los módulos cliente los referencian desde la parte superior de la jerarquía y como clientes cuando referencian módulos de la parte inferior.

En Linux, los módulos se pueden cargar y descargar con *insmod/rmmod (modprobe* resuelve las dependencias al cargar un módulo). Con módulos apilables, podemos definir dependencias entre módulos, lo que evita la duplicidad del código y la descarga y carga innecesaria de módulos que se usen frecuentemente.
Los módulos se organizan en una lista enlazada que comienza por un pseudomódulo.
Cada módulo se define mediante dos tablas: la de **módulos** y la de **símbolos**. La tabla de módulos está formada por:

| | |
|---|---|
| ***next:** puntero al siguiente módulo. | **Size:** tamaño del módulo en páginas de memoria. |
| ***name:** puntero al nombre del módulo. | **Usecount:** contador de uso del módulo. Se incrementa cuando comienza una operación y se decrementa cuando finaliza. |
| ***syms:** puntero a la tabla de símbolos. | **Nsyms:** número de símbolos exportados. |
| ***deps:** puntero a la lista de módulos que referencia este módulo. | **Ndeps:** número de módulos referenciados. |
| ***refs:** puntero a la lista de módulos que usa este módulo. | **Flags:** opciones del módulo. |

La tabla de símbolos define aquellos símbolos controlados por este módulo que se utilizan en otros lugares.

Los principales componentes del núcleo del sistema son:

- **Señales:** el núcleo las usa para llamar un proceso. Por ejemplo, se pueden utilizar para notificar ciertos fallos a un proceso, como una división por 0.
- **Llamadas al sistema**: es la forma en la que un proceso requiere un servicio específico del núcleo. Las llamadas al sistema se agrupan en seis categorías: *sistema de ficheros, proceso, planificación, comunicación entre procesos, socket (red) y misceláneas.*

Los módulos comienzan con las *init_module* y *delete_module* que se localizan al inicio y al final del archivo del módulo y se encargan de cargar y descargar el módulo. Se puede utilizar *request_module* para cargar también los módulos dependencia necesarios.

Los **namespaces** proveen al usuario de una forma ligera de virtualización permitiendo contemplar las propiedades globales de un sistema bajo distintos puntos de vista. Tradicionalmente, muchos recursos se han manejado de forma global en Unix (por ejemplo, los procesos se identifican por su PID, por lo que debe haber una lista global de PIDs gestionada por el kernel). Pasa algo similar con los usuarios y sus UID. Para aislar al usuario de una vista tan global del sistema, surgen los *namespaces*, virtualizaciones ligeras que aíslan distintas partes del Sistema. Esto tiene varias ventajas, entre ellas dar al programador una visión más local del servicio que quiere inspeccionar y evitar que una modificación errónea corrompa otros servicios, ya que los namespaces son independientes . Los procesos que se ejecutan en distintos namespaces son totalmente independientes, pudiendo llegar a coincidir sus PID ya que estarían ejecutándose sobre máquinas distintas debido a la virtualización. Los namespaces pueden mantener una relación de jerarquía. Cada namespace tiene su padre y cada padre puede tener varios hijos.
Al crear un proceso con *fork* o *clone*, podemos decidir si se crearán en un nuevo namespace, se ejecutarán en el namespace del padre, etc.

Los **control groups** son una herramienta que ofrece el kernel y permiten crear colecciones de procesos que pueden organizarse de forma jerárquica. Permiten una gestión eficiente de la asignación de recursos a determinados procesos. Permiten el uso de prioridades (*nice).*