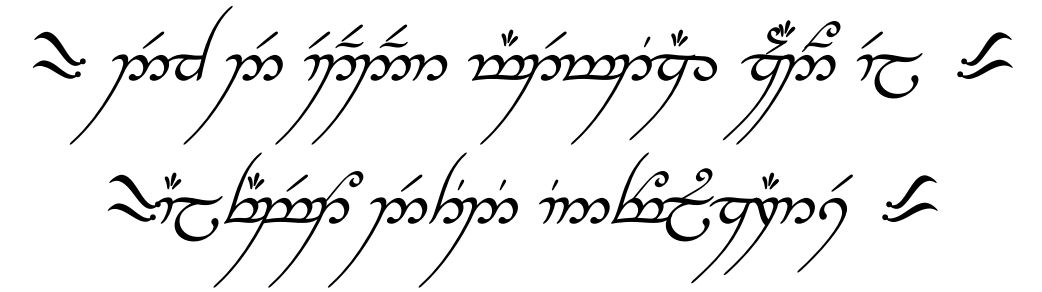
Escuela Técnica Superior de Ingeniería Informática

Asignatura:
Sistemas Operativos

Autor:
Fernando José Mateos Gómez

Ultima Modificacion: **2 de Febrero del 2022**



Indice

1. Tema 1: Fundamentos

1.1. Sistema Operativo

El *Sistema Operativo* es un software que facilita el uso del sistema informático, "SI", enlazando al usuario con el hardware, mediante las **interfaces entre las aplicaciones y el sistema** y las **interfaces entre el usuario y el sistema**.

1.1.1. Kernel

El kernel o también llamado "núcleo del sistema", es una capa situada "por encima", un nivel de abstracción intermedio entre el *hardware y las herramientas* que hacen la función de *interfaces*, estas herramientas son de 3 tipos:

- **Herramientas del Sistema**: Intérprete de comandos, administración del sistema, CLI ...
- **Herramientas de Desarrollo**: Compiladores, depuradores, IDE's ...
- **Aplicaciones**: Navegador, juegos ...

Linux no es un SO, sino un Kernel

1.2. Procesos e Hilos

Los *procesos* son objetos del "SO" que encapsulan un número determinado de *hilos* en un espacio de memoria, y cuenta con recursos de **hardware y sofware**.

Gracias al *kernel*, creamos una capa de abstracción, por la cual definimos un espacio de memoria propio para cada proceso, que no puede ser usado o visto por otro proceso, ya que cada espacio de memoria está definido para un solo proceso.

Es decir, **los procesos están incomunicados entre si**, sin embargo la cooperación entre procesos se debe al propio kernel, que proporciona herramientas, y además un proceso es capaz de crear otros procesos y acceder al sistema, unicamente solicitando servicios.

En Windows, el espacio estandar que se reserva por cada proceso es de 2Gi

1.2.1. Multiprogramación

Considerando que tenemos varios procesos cargados, y que alguno está bloqueando al sistema (*requiere una cantidad de tiempo muy superior al tiempo que puede cargar instrucciones el procesador*), podemos aprovechar la existencia de varios procesadores y ejecutar varios de estos procesos en cada procesador, aumentando la velocidad.

Podemos suponer que cada procesador ejecutará sus procesos de forma independiente al resto, sin embargo en la realidad, usaremos un **planificador**, *que en vez de planificar procesos, planifica hilos*, que liberará un procesador para que otro "trabaje", para esto el planificador usará algoritmos de planificación tales como el *FIFO*, *LIFO* ...

Estos algoritmos, trabajan siempre bajo la misma premisa:

1. El proceso está **preparado**, este está buscando un procesador para poder ejecutarse y se mantendrá en esta etapa mientras el planificador lo indique.

2. Cuando encuentra un procesador, el proceso se pone en **activo**, y se ejecuta.
3. Al terminar el proceso, se queda **bloqueado**, deja de consumir tiempo del procesador y se mantendrá así hasta que se requiera volver a usarlo y volverá a estar **preparado**
4. De forma intermedia, podemos **apropiarnos** del procesador en medio de la ejecución del proceso, lo que lo llevará a estar **preparado**

El tiempo que tarda un proceso, no tiene relación con el tiempo que consume del procesador, por lo que las operaciones que dependan del sistema, se harán a través de herramientas del sistema.

1.2.2. Hilos

Denominamos a los *hilos* como secuencias de procesos que comparten el mismo espacio de memoria.
Estos nacen debido a la *paralelización* de procesos, cuando tenemos varios procesadores podemos intentar ejecutar varios procesos a la vez, sin embargo si estos procesos dependen del estado de los otros, al no poder compartir recursos entre ellos, usamos un espacio de memoria reservado para un conjunto de procesos, de forma que ahora pueden comunicarse entre si. Solo no es rentable cuando se ejecutan operaciones de cálculo.

1.3. Conceptos básicos sobre Hardware

1.3.1. Organización básica de un Ordenador

El *ordenador* es un conjunto de procesadores, unidos a una memoria y a diversos controladores, que a su vez están conectados a un bus de datos.
Cada procesador accede a su parte de la memoria, **no pueden leerse y escribir a la vez**, pero al resto de la memoria que no está asignada a los procesadores, se podrá acceder de esta forma simultaneamente.

1.3.2. Interrupciones

Es una alteración en la secuencia de ejecución de las instrucciones del procesador. Tiene distintas causas:

- **Interrupción del *Hardware***: Como desconectar algún controlador.
- **Excepciones**
- **Ejecución de instrucciones de petición de interrupción**: Como *INT* o *TRAP*

Las *interrupciones* las tratamos de la siguiente forma:

1. Termina de ejecutar la CPU la instrucción actual.
2. Almacenamos el estado del procesador en un registro.
3. La CPU pasa a modo *supervisor*
4. Determinamos la dirección de una *Subrutina de Servicios de Interrupción*, "SSI", que es una tabla de vectores que almacena codigos de interrupciones.
5. Retornamos el valor de la tabla de SSI y restauramos el estado anterior.
6. Continuamos con la ejecución del proceso.

1.3.3. Excepciones

No son más que errores que suceden durante la ejecución de una instrucción.

1.3.4. Modos de Ejecución de un procesador

El procesador ejecutará una serie de *instrucciones*, operaciones aritméticas; lógicas o de movimiento de datos.

Podemos diferencias dos tipos de modo de ejecución:

- **Supervisor**: Con este modo, el procesador puede ejecutar cualquier instrucción. Para pasar de *supervisor* a *usuario*, no es necesaria cumplir ninguna condición.
- **Usuario**: Solo puede ejecutar las *instrucciones no privilegiadas*, aquellas que no accedan a recursos del *hardware* ni recursos del *software* externos al actual. Para acceder al modo *supervisor*, es necesaria la ejecución de una interrupción, y esto ocurrirá si y solo si, cuando el proceso que quiera ejecutar pertenezca al *kernel*.

1.4. Memoria

La memoria funciona como un array de bits, guardando en direcciones consecutivas, podemos acceder a través de un índice que empieza desde el 0, y puede transmitir en potencias de 2 hasta 8 bytes.

1.4.1. Paginación

Es una técnica que se implementa por Hardware, a través del "MLU" que divide las direcciones lógicas en partes, a través de un *traductor* que accede a una tabla *de paginación*.

La memoria se reordena al usar este traductor, dividiendo la memoria en dos partes:

- **Página** pasa a ser **marco**, que almacena la dirección física, e indica la dirección donde se guarda la información.
- **Desplazamiento**, se mantiene igual pero solo puede tener un tamaño de 2^kbits.

1.4.2. Mapa de Bits

Esto es la tabla de traducciones, que asocia el contenido de cada marco con una página. Dentro de este mapa existen ciertos bits, que informan sobre el estado y contenido de este mapa. Entre estos destacamos el bit P que si vale cero, indica que el marco no está asignado. Se usa para poder usar un proceso sin necesidad de que tenga todos los marcos cargados, a lo que denominamos **Memoria Virtual**.

1.4.3. Memoria Virtual

Por defecto, todos los marcos no están cargados, lo que lanzará una excepción y enviará la página a un marco libre. Si no hay marcos libres, se ejecutará un algoritmo para verificar qué pagina no está en uso, a futuro, y la cargará en este nuevo espacio.

1.5. Acceso a Dispositivos

Para los SO, los dispositivos no son más que cajas negras que pueden transmitir información, reciben órdenes e informan sobre su estado.

Existen los *controladores*, interfaces electrónicas del dispositivo, que controla el funcionamiento y recibe órdenes del procesador. Podemos crearlos específicos para un dispositivo.

1.5.1. Adaptadores

Son dispositivos que sirven como interfaces para comunicar un dispositivo con otro. Esta interfaz proporciona:

- **Características mecánicas**: Como el número de pines.
- **Características eléctricas**: Como la velocidad o el número de bits.
- **Protocolos**

Podemos acceder a estos dispositivos a través de un mapeo de de los registros, buffers de lectura/escritura, que lanzaran interrupciones acorde al número de dispositivos que haya. De esta forma informan de los cambios de estado a través de:

- *Moficando el valor del registro*
- *Con interrupciones*

Y se gestionen mediante sondeos, que no deben de hacerse con procesadores, y como ya hemos dicho, interrupciones.

1.5.2. Arquitectura I/O

- **Driver**: Es un componente del Kernel, que controla dispositivos.
- **Sistema de Archivos**: Crea una capa de abstracción de un array de sectores, que permite el uso de archivos en cualquier lugar.
- **Biblioteca de Funciones**: Almacena API's y pequeños programas concretos.

1.6. Unidades de Almacenamiento

1.6.1. Unidades Magnéticas

A través de un adaptador interfaz "SATA", "USB", "SAS"... se conecta a un adaptador controlador, que se conectará a la RAM, volatil, y a un almacenamiento por placas magnéticas, permantente.

Solo transmite información por sectores (dividido en placas ferromagnéticas y pistas concéntricas, cada cara puede leerse y escribirse), y es muy lento.

$$\boxed{T_a = T_m + nT_c + T_h + T_r + T_x}$$

Siendo:

- T_m es el tiempo de arranque.
- nT_c es el número de pistas a saltar por un factor de tiempo medio para saltar entre cada pista.

- T_h es un factor de inercia.
- T_r tiempo de rotación.
- T_x tiempo de lectura por sector.

Existen ciertas optimizaciones como por ejemplo, *transferir bloques consequitivos*, reduciendo el tiempo de lectura notablemente, *leer los sectores en el orden que pasan* o *reservar bloques en caché*.

1.6.2. SSD

Sustituye al disco magnético, en vez de usar placas magnéticas usa un array de memorias flash.

Al hacer operaciones de escritura o lectura, debemos de considerar un factor, cada 100.000 borrados la celda flash debería de modificarse, porque no borra completamente su contenido. Y ya que al borrar información, debemos de borrar una página de memorias flash, debemos de tener en cuenta que los datos persistentes que no deberían de borrarse, se almacenan en un buffer temporal y se vuelven a escribir. Se usan ciertas optimizaciones para evitar tener que cambiarlas continuamente:

- **Wear Leveling**: Hay sectores que se escriben con mucha frecuencia, por lo que usando un Wear Leveling *dinámico*, establecemos una relación entre las direcciones físicas con las lógicas, de forma que no reescribe en las direcciones lógicas directamente sino en las físicas, evitando escrituras masívas.

  Por otro lado, el *estático*, hace lo mismo, pero reestructurando los sectores, los mueve.
- **Trim**: Un comando implantado en el Kernel (Linux 2, Windows XP, ...) que resuelve el problema de tener que borrar y escribir un sector inmediatamente, haciendolos irrelevante, por lo que si son borrados no ocurre nada, y la actualización y borrado son más rapidos.

1.7. Interfaz de Usuario

Podemos catalogarlos en dos tipos:

- **Terminales Externos**: A través de un emulador, nos conectamos a otro dispositivo.
- **Consolas**: Dispositivos que se conectan directamente con la máquina y permiten su interacción directa con él. Se conforma por 3 componentes independientes: *Pantalla*, *teclado*, *ratón*.

1.7.1. GPU

Siendo este el *adaptador de pantalla*, guarda información en memoria, el procesador o la memoria principal (siendo esta la más común), tras analizar el estado de la pantalla, *memoria de vídeo*.

Dependiendo de como se use esta memoria, se puede analizar la información de dos formas (relacionadas con la calidad que se quiera presentar por pantalla):

- **Modo Texto**: Se guarda el color, brillo, opacidad... de los caracteres, a partir de un array de caracteres.
- **Modo gráfico**: Se escanea un array de pixeles, y se guarda información sobre el RGB, código que expresa colores.

1.7.2. Teclado

Posee integrado un microcontrolador capaz de exlorar el array de caracteres del teclado varias veces por segundo, almacenando información sobre el estado de las teclas (pulsadas o levantadas), guardando entonces eventos al teclear o usar atajos de teclado.

1.7.3. Ratón

Funciona igual que el teclado, posee un microcontrolador que genera un evento por cada click que se hace, y otro por cada vez que se cancela el click, se levanta el botón.

En vez de guardar información sobre el tipo de tecla, como el teclado, se guarda información del ángulo de un vector posición, que se mide en "Mickies".

1.8. Arranque del Sistema

Todos los componentes "hardware" del sistema se deben de inicializar en el arranque, y siguen el mismo procedimiento:

1. El procesador genera un proceso de inicialización que se transfiere a una dirección de memoria determinada.
2. Si el sistema operativo, no se encuentra en ROM, llama al *cargador de hardware*, sino, irse al punto 5.
3. El *firmware* o cargador de hardware (UEFI o BIOS), realiza un encendido mínimo y verifica controladores conectados, además de leer el *cargador de software*.
4. El cargador de software carga el kernel y le transfiere el control una vez cargado.
5. Se inicializa el kernel y realiza ciertas funciones:
   a) Crea procesos de sesión y estructuras de datos.
   b) Checkea el sistema.
   c) Carga dispostivos en el kernel.

1.9. Terminología de Redes

- **Adaptador de red**: Dispositivo que permite conectar el ordenador a la red.
- **Dirección MAC**: Dirección única que permite identificar al adaptador de red en una red local.
- **Switch**: Dispositivo que permite conectar varios segmentos de red en un punto, permitiendo controlar el tráfico de red.
- **Router**: Dispositivo que es capaz de conectar varias redes.
- **Gateway**: Dispositivo que se usa para conectar redes con distintos protocolos.
- **Dirección IP**: Dirección que identifica a cada ordenador de una red. Existen dos tipos, *IPv4* e *IPv6*, pero trabajaremos con la primera.

  Se usa el formato "CIDR", $xxx.xxx.xxx.xxx/n$, siendo xxx un número de 3 dígitos y n el número de bits usados para la máscara de red. Hay hasta $32 - n$ bits para los hosts, que deben de estar a cero.

  Además podemos catalogar las redes IPv4 en 3 tipos:

- **Clase A**: 24 bits para los hosts, (10.0.0.0 → 10.255.255.255).
- **Clase B**: 20 bits para los hosts, (172.16.0.0 → 172.31.255.255).
- **Clase C**: 16 bits para los hosts, (192.168.0.0 → 192.168.255.255).

1.10. Conceptos Básicos de SSOO

1.10.1. Llamadas al Sistema

No son más que peticiones que se hacen al núcleo del SO par aobtener algún servicio, a través de un proceso. Normalmente lo vemos representado como las API, conjunto de llamadas al sistema que ejecutan programas, que se pueden implementar a través de:

- *Interrupciones*: Se accede mediante una interrupción, que pasará el sistema a modo supervisor, y podremos acceder a una dirección de memoria en función del hardware. Finalmente se restaurá el estado previo a la llamada.
- *Instrucciones Específicas*: No importa la dirección de entrada y se ejecutan directamente en modo supervisor, sin embargo debe de hacer todas las comprobaciones que haría el modo supervisor.
- *Rutinas*: No son factibles, debido a que es complicado calcular la dirección de memoria que estarían las rutinas y no pueden pasar al modo supervisor.

1.10.2. Usuarios

No son más que personas autorizadas a usar el sistema, que determinarán que derechos tendrán sobre un proceso que acceda al sistema.
Se les identifica con un UID, y si se agrupan en grupos, con un GID

1.10.3. Archivos

Conjunto de información guardado, estructurado de forma jerárquica en directorios, que residen en dispositivos.

1.10.4. Intérprete de Comandos

Programa interactivo que lee comandos del usuario via terminal y lo traduce a llamadas del sistema.

1.10.5. Interfaces Gráficas de Usuario

Programa interactivo que realiza la misma función que un intérprete de comandos pero de forma gráfica, GUI.

1.11. Modelos de Diseño

1.11.1. Modelo Monolítico

Todo el sistema se encuentra alojado en un único espacio de memoria, que no desorganizado. Por lo que todos los procesos y estructuras de datos son accesibles y manipulables entre si. Usa un módulo que controla los servicios, llamado **despachador**, una única rutina que hace las llamadas. Linux es monolítico.

1.11.2. Modelo en Estratos

Capas separadas para generar una mayor capa de abstracción, resolviendo cada una tareas específicas. Entre el usuario y el Hardware distinguimos 4 capas, llendo de mayor a menor nivel de cercania al hardware, tenemos:

- **Capa 0**:Proporciona Hardware Multiprogramado.
- **Capa 1**: Dota a cada proceso de su propio estado de memoria.
- **Capa 2**: Dota a cada proceso de su propia consola virtual.
- **Capa 3**: Cada proceso es capaz de usar dispositivos de entrada y salida sobre otros.

Vemos que es facil de depurar pero perdemos la compartición de memoria.

1.11.3. Modelo Micronúcleo

Dividimos el nucleo en sectores, manteniendo las interrupciones y la comunicación entre procesos en el *microkernel* y el resto de funcionalidades en modulos externos, menos aquellas funcionalidades que requieran de multiprogramación.

Ofrece una gran robustez y facilidad de depuración, pero es extremadamente lento. El único SO, que ha logrado implementar este modelo es el descontinuado Minix.

1.11.4. Ejemplos

- En el caso de **Linux**, al ser Monolítico, requiere de su gran comunidad para mantenerse, y su núcleo carga los módulos necesarios para realizar las tareas, por lo que es extremadamente rápido.
- **Windows** es también Monolítico, y separa los procesos de aplicaciones del propio kernel, sin embargo manteniendo las aplicaciones de usuario en el lado de los procesos de servicio.

2. Tema 2: Sistemas de Archivos

2.1. Funcion de los Sistemas de Archivos

La intención de un sistema de archivos, no es más que la de ser capaz de implementar archivos y directorios sobre la estructura de sectores de las unidades, con la capacidad de leer y modificar estos.

Hay varios servicios que se es capaz de usar:

- Modificar sobre archivos enteros.
- Modificar el contenido de archivos.
- Crear, montar o borrar archivos en otras unidades.
- Protección y compartir archivos.

2.1.1. Particiones

Podemos dividir una unidad física en unidades lógicas, y existen dos tipos de **tablas de partición**:

- **MBR**: Usada por la *BIOS*, usa 4 particiones, 3 primarias y una extendida, y admite particiones de hasta 2TB.
- **GPT**: Común de la *UEFI*, permite hasta 128 particiones de hasta 2ZiB cada una.

2.1.2. Volumenes

No son más que las unidades lógicas en las que dividimos una partición, y ocupan una partición e incluso hasta una unidad física entera.

En función del sistema operativo se accede de forma distinta, en *Windows* mediante una letra asignada a la unidad, y en *UNIX* a través de un árbol de directorios.

Está compuesto, típicamente, por un cargador del SO, una estructura para gestionar el espacio libre o los bloques defectuosos y un directorio raiz.

2.1.3. Directorio

Es una estructura de datos del sistema de archivos que contiene información sobre los archivos en su interior. Poseen una estructura jerárquica y guardan información como el UID y el GID, su nombre, fechas de creación; modificación o ultimo acceso, su tamaño, etc...

Como todo volumen posee al menos un directorio, aquel que engloba a todos los demás se le denomina *directorio raiz*.

2.1.4. Archivos

Una secuencia de bytes que se pueden leer o modificar, bloque a bloque o uno a uno.

2.1.5. Gestión del Espacio

La información no se asigna sector a sector, sino en bloques, así la gestión es más eficiente. Sin embargo esto genera un problema, ¿cómo saber cuales son los bloques ocupados?, pero se soluciona con un gestor de espacio que almacena los bloques no ocupados y los separa de los otros, o con un *mapa de bits* que indica el estado del bloque.

En caso de que haya bloques defectuosos, no debemos de asignar archivos ahí, por lo que los agrupamos todos juntos y los separamos de los demás.

2.2. FAT

Este es el sistema de archivos más simple posible, es antiguo y lo usan principalmente memorias USB.

2.2.1. Organización

Dividimos el volumen en 5 bloques:

- **Sector 0**: Se encarga del arranque del sistema, cargador de software con un MBR.
- **Sector 1**: Contiene información útil para acelerar ciertas operaciones.
- **Copia 1**
- **Copia 2**
- **Bloques**: Amacena los bloques del directorio raiz y los defectuosos, en las dos primeras posiciones, el resto están libres.

2.2.2. Contenido

Cada bloque del disco tiene una referencia en una tabla, esta entrada en la tabla tiene 3 posibles valores (FREE, BAD, EOF), indicando si está libre, defectuoso o si es el último bloque del archivo.

Guardamos la siguiente información:

- Nombre del archivo: 8 bytes
- Extensión: 3 bytes
- Atributos: 1 bytes
- Reservado: 10 byte
- Hora de modificación: 2 bytes
- Fecha de modificación: 2 bytes
- Número del primer bloque: 2 bytes
- Tamaño archivo: 4 bytes

2.2.3. Acceso

Consideremos un fichero que se llama F1 y guarda bloques en los espacios 3, 10 y 12.

En la tabla de referencia, guardamos el primer bloque (con valor 3), tras esto nos movemos a la posición 3 en la memoria y guardamos el valor 10, finalmente nos movemos a la posición 10 y guardamos el valor 12, como no quedan más direcciones de memoria a las que asignar bloques, la posición 12 queda marcada como EOF.

Para transferir una posición N de un fichero, debemos de saber el tamaño del bloque (4KB), y calcularemos el bloque lógico:

$$\boxed{B_L = \frac{N}{4KB} = X}$$

Este valor X indica la posición en el bloque físico, el valor que hemos almacenado ahí. Para indicar los sectores a transferir deberíamos entonces sumar el bloque físico más el consecutivo en la dirección todo respecto a S, el primer sector en bloque:

$$\boxed{S + 2 * M[X] + S + 2 * M[X] + 1}$$

2.3. Ext2

Estructura usada por los sistemas UNIX

2.3.1. Organización

El volumen es organizado en una estructura de $N + 1$ bloques, siendo el primero el del sector de arraque y el resto son grupos de bloques, que cada uno almacena una serie de información en bloques más pequeños:

- **SuperBloque**: Contiene la versión del bloque, el número de veces que se ha montado, un número mágico (EF53) y el tamaño de los bloques consecutivos.
- **Descriptores**: Contiene las direcciones donde comienzan el resto de bloques siguientes.
- **Mapa de Bits**: Guarda el estado de cada archivo
- **Mapa de Bits Nodos-I**
- **Tabla de Nodos-I**: Contiene la misma información que un FAT, pero con añadidos, el UID y GID del propietario, el número de enlaces, número de bloques indirectos
- **Bloques**: Guardan información como el nombre del registro, ya que almacena información sobre los directorios, y sus nodos-i.

2.3.2. Contenido

Para aumentar su capacidad, utiliza tablas de enlazamiento indirecto, hasta 3 veces, por lo que a parte de generar un mayor espacio, si existen N bloques, habrán N^3 bloques generados por el direccionamiento indirecto, al coste de un tiempo más lento para encontrar archivos.

2.3.3. Acceso

La forma de direccionar archivos, es el mismo que con FAT.

2.3.4. Enlace Directo y Simbólico

Hemos mencionado antes que se guardan contadores con información sobre el número de enlaces, que no es más que links, que podemos crear para duplicar un archivo sin necesidad de que ocupe el doble de memoria, sin embargo esto puede generar problemas, ya que borrar el archivo principal rompe cualquier enlace posterior.

Con este motivo, se crean los enlace simbolicos

2.4. Reserva

Se le denomina así a la caché del disco, no es más que un espacio de la memoria de facil acceso, rápido, usando tablas de hash

2.4.1. Modos de Reserva

- **Escritura Directa**: El bloque se actualiza inmediatamente tras la escritura, sin embargo no se beneficia de la reserva.
- **Escritura Diferida**: Se actualiza una copia en memoria, y no podemos decir con seguridad la seguridad.

2.5. RAID

Usando varios discos en paralelo conseguimos simular un disco mejor. Al ser un array de discos, tendremos ciertos problemas a resolver, pero también podemos utilizar paralelización para aumentar su velocidad.

Podemos implementarlos mediante hardware o software.

2.5.1. RAID 0

Es el más simple de todos, y se basa en el uso de N discos, apilados de forma consecutiva, de forma que la información la podemos dividir en sectores de K bloques.

Es bastante rápido, sin embargo si uno de las pilas se rompe, el sistema cae completamente.

2.5.2. RAID 1

Es una mejora del anterior, no aprovecha la velocidad de leer en paralelo, pero hay mayor seguridad al apilar los los bloques, y copiar los datos de uno a otro.

2.5.3. RAID 4

Hay N pilas, y una extra, se le llama disco de paridad, y guarda una coppia de cada sector.

2.5.4. RAID 5

En vez de haber una pila extra, los sectores del disco de paridad se distribuyen aleatoriamente a lo largo de los discos.

2.5.5. RAID 6

Usa el algoritmo “Reed-Solomon” para detectar y corregir errores.

2.6. COW

Método de copia, que se basa en almacenar una copia en el mismo espacio de memoria lógico. Es aplicable a toda clase de estructuras de almacenamiento, *volúmenes* o *archivos*.
De forma general ocurre lo siguiente:

- La estructura queda bloqueada.
- Se genera un COW.
- Si...
  - Se *lee*, se busca en el COW, si no está actualizada, en la estructura original.
  - Se *escribe*, se hace sobre el COW.

Un COW puede eliminarse, lo cual descartará cualquier cambio realizado, pero también fusionarse con la estructura con el fin de guardar los cambios permanentemente.

2.6.1. Archivos

Usamos tablas de bloques para gestionar el espacio, por lo que solamente sabiendo el índice podemos saber donde está guardado el archivo y su COW, y como comparten un mismo espacio de memoria podría dar lugar a problemas de *incoherencia*. A este sistema lo solemos denominar como **bloques sombra**.
Al escribir sobre el archivo, seguirá haciendolo sobre el COW y la lectura también. En caso de **deshacer** los cambios, será necesario borrar la información referente en los nodos-i y los bloques indirectos, si se desea hacerlos **permanentes**, se borra el bloque original y se asigna al COW.

2.6.2. Usuarios

A nivel de usuario, en vez de usar tablas, lo que hacemos es almacenar un identificador con el archivo, una clave, lo que nos permitirá buscar el archivo rápidamente. En caso de añadir archivos o modificarlos, crearemos una COW, que no se hará persistente hasta que se lo indiquemos. En caso de hacerlos persistentes, fusionamos el COW con el original y borraremos el COW, después de hacer el cambio. Si se desea deshacerlo, solo debemos de borrar el COW.
Los COW se guardan en otro archivo.

2.6.3. Volúmenes

Funciona mediante el uso de claves y es exactamente igual que con los usuarios, con la diferencia de que los COW se guardan en otro volumen.

2.7. Journaling

En caso de que el sistema se caiga en mitad de una operación, el sistema de archivos podría quedar insconsistente, por lo que debemos de tener algún sistema por el cual podemos "reconstruir" el sistema de archivos.
Como solución agruparemos todas las operaciones en una **transacción**, donde si falla una operación se cancelan todas. Podemos implementarlo mediante COW o con logs.

ANTES de cada transacción, crearemos un archivo ".log", donde guardaremos el estado del valor original, para evitar que en caso de que se cancele o falle, podamos recuperarlo sin posibilidad de perder datos o no poder restablecer los cambios.

Miraremos el valor de log, con el fin de indicar si se ha realizado correctamente o no. Si no ha fallado, se borra el log, sino, usaremos el log y restableceremos el estado del sistema.

2.8. Sistema de Archivos Distribuidos

No es más que el concepto de usar dispositivos en la nube o en servidores para almacenar datos.

Las principales unidades remotas son:

- **Unidades de Bloques**: Existe un servidor que funciona como una unidad física, que recibirá peticiones a nivel de driver, y poseerá una estructura de archivos decidida por el cliente.
- **Sistemas de Archivos Lógicos**: Un servidor creará una estructura de archivos, y recibirá peticiones a nivel de administrador de archivos.

2.8.1. Unidades de Bloques

El nivel de administración se encuentra del lado del cliente y se comunicará con un gestor de disco virtual que simulará uno real. Se permitirá operaciones de escritura y lectura simultaneas, siempre y cuando no se realicen sobre el mismo registro.

2.8.2. Sistemas de Archivos Lógicos

La administración queda completamente del lado del servidor, el cliente hace poco.

2.8.3. Virtual File System

Conocido simplemente como VFS, es una forma de implementar distintos sistemas de archivos en una máquina y puedan convivir. Se usa un sistema de archivos virtual, el cual, en función de las necesidades del usuario cargará la estructura de archivos que este quiera, para un volumen concreto.

2.9. Autentificación de Usuario

Mediante el uso de claves o identificadores, buscamos la seguridad de los usuarios. El método más interesante a estudiar es el "AaaS", usando un sistema físico externo, o el que RSA, el cual se basa en generar dos claves (Pública y Privada) con el fin de que una sola persona pueda codificar (Pública) y solo el receptor pueda decodificarla (Privada).

2.10. Control de Acceso

En el momento en el que el usuario queda registrado debemos de encontrar alguna forma por la cual sepamos a que recursos esté permitido acceder. Como los usuarios ejecutan instrucciones sobre "objetos", a estos derechos los llamaremos **derechos de acceso**, que se aplican a la operación sobre un objeto, o **dominio de protección**, el cual abarca al conjunto de derechos de acceso de un usuario. Estos dominios son transferibles, modificables y solapables con otros.

2.10.1. Matriz de Control de Acceso

Montando una matriz doble, por la cual guardamos los permisos de cada dominio por archivo, y sobre cada dominio por los demás, somos capaces de solventar este problema, sin embargo es inpracticable debido al tamaño y los recursos que tomaría.

Como solución alternativa podemos utilizar una lista abreviada, usando comodines.

3. Tema 3: Virtualización

3.1. Conceptos Previos

Un hipervisor no es más que un software que se implementa entre la capa de hardware y la del Sistema operativo, la cual simula un sistema operativo, o varios, con su propio hardware.

3.2. Hipervisores tipo I

Lo ejecutamos directamente sobre un anfitrión, por lo que simulan memoria de video, dispositivos y procesadores, simulando un espacio físico virtual.

En este tipo, el hipervisor es el que coge un fragmento de la memoria del anfitrión y recoge los eventos del teclado y ratón con el fin de poder simular el comportamiento de una máquina real.

Podemos simularlos de dos formas:

- **Estático**: Guardará el espacio ocupado por igual en físico que en virtual.
- **Dinámico**: Ocupa lo necesario, pero es más lento y requiere de indices para acceder a archivos compartidos.

En el caso de la red podemos utilizar "adaptadores puente" simulando una tarjeta adaptador de red, con su propia IP y MAC, o una "NAT", con una IP privada que se obtiene mediante un router simulado.

En este tipo, el procesador se simula, de forma que las instrucciones que requieran de ser privilegiado, serán enviadas, en función del estado del usuario en ese momento, pero todo desde el *hipervisor*

- **Si ya estaba en modo supervisor**: Lo que mandará será una excepción al hipervisor y este ejecutará la instrucción.
- **Si no lo estaba**: Se producirá una interrupción que le cederá el control al hardware del sistema anfitrión, que le cederá el control posteriormente al hipervisor.

Por lo que podemos ver, entonces el procesador tendrá dos modos, *no root* y *root*, que se cambiarán para ejecutan las instrucciones de acorde a los casos anteriores.

Además, cada hipervisor viene creado con una estructura de control llamada "VMCS", que es capaz de guardar el estado de la máquina en todo momento, indicar que dispositivos están conectados a esta (*device passthrough*, adaptadores, GPU, etc...) y los simulados. Además de poseer un **IOMMU**, una unidad de manejo de memoria que conecta un bus de IO con la memoria principal.

El proceso para ejecutar un hipervisor es el siguiente:

- El hipervisor se inicia.
- Se crea un VMCS por cada máquina virtual.
- Se lanzan las máquinas virtuales.
- Se transfiere el control al hipervisor por cada instrucción privilegiada.
- Se cierra el hipervisor.

Para concluir, podemos decir que esta clase de hipervisores deben de instalarse sobre procesadores que permitan *virtualización por hardware* o que cualquier instrucción privilegiada en modo usuario, lanze una *excepción*.

3.3. Hipervisores tipo II

Al contrario que la anterior, este tipo de hipervisores se ejecuta sobre el SO anfitrión, en conjunto con procesos de usuario. Imita todo lo que hace el hipervisor anterior salvo en la *simulación del procesador*.

En este modelo, las instrucciones **se ejecutan una a una**, y el hipervisor analiza previamente la secuencia de instrucciones hasta encontrar una instrucción condicional, dependiendo de las instrucciones que contenga se hará lo siguiente:

- **Privilegiadas**: El hipervisor ejecuta estas instrucciones.
- **No privilegiadas**: Se ejecutan directamente.

3.3.1. Ventajas y Contras

Comparandolo con el anterior, este es más *ineficiente* ya que analiza más instrucciones pero es más *flexible*.

3.4. Paravirtualizadores

Estos hipervisores se ahorran cualquier problema con el procesador, y lo sustituyen por una API que llamará al sistema anfitrión, pero para que esto ocurra el SO invitado debe de estar compilado previamente, ya que si no, no podrá analizar todas las instrucciones previamente.

Como solución, se implementa una capa de abstracción, una interfaz entre el kernel y el hardware (VMI), que funciona de forma independiente.

Es bastante eficiente, tanto como una máquina física, pero no ofrece transparencia, gracias a su API.

3.5. Hipervisores Hibridos

Hoy en día se combinan los tres tipos, ya que casi todos los SO son *monolíticos*. Se basan en una implementación de un **Hipervisor del tipo II**, y como casi todos los dispostivos soportan virtualización en el hardware, ya tendremos el **hipervisor de tipo I** implementado. En cuanto a los **paravirtualizadores**, aprovechan que las marcas crean drivers y módulos y por ende podemos aprovechar sus api para el sistema.

4. Tema 4: Contenedores

4.1. Dificultades en el Despliegue

Comparando una máquina virtual con un socket, podemos observar un problema fundamental: **una máquina virtual tiene que virtualizar desde el hardware hasta sus aplicaciones, mientras que un socket solo las aplicaciones**.

Usar varias máquinas virtuales es una solución, pero evidentemente no es factible, ocuparían mucho más espacio.

4.2. Contenedores: Sockets

Guardan en un repositorio las aplicaciones que necesite, y las máquinas que los vayan a ejecutar, tendrán programas capaces de desplegarlos, cada uno de forma aislada.

Docker, *Podman* o *Linux Containers* son opciones viables.

4.3. Escalabilidad y Tolerancia a Fallos

4.3.1. Escalabilidad

Propiedad que indica la capacidad que tiene el sistema de ampliarse o decrementarse sin muchos cambios. Solo podemos hacerlo, al *replicar aplicaciones sin estado*, que no tengan datos o si lo están, estén guardados de forma externa.

4.3.2. Tolerancia a Fallos

Propiedad que indica la robustez que tiene un sistema a cambios, sin posibilidad a errores. Si podemos desplegar un contenedor, sin romper el resto, será que tenemos un sistema robusto.

4.4. Servicios en la Nube

Hoy en día hay distintos tipos de implementaciones, **sobre hardware**, **hardware como servicio**, **infraetructura como servicio**, **plataforma como servicio** o **software como servicio**.