**FILE SYSTEM**

**¿Porque necesitamos archivos?**

* Almacenar grandes cantidades de datos
* Tener almacenamiento a largo plazo
* Permitir a distintos procesos acceder al mismo conjunto de información

**Archivo**

* Entidad con nombre
* Espacio lógico continuo y direccionable
* Provee a los programas de datos (entrada)
* Permite a los programas guardar datos (salida)
* El programa mismo es información que debe guardarse

**Archivos - Punto de vista del usuario**

* Qué operaciones se pueden llevar a cabo
* Cómo nombrar un archivo
* Cómo asegurar la protección
* Como compartir archivos
* No tratar con aspectos físicos
* Etc.

**Archivos - Punto de vista del diseño**

* Implementar archivos
* Implementar directorios
* Manejo del espacio en disco
* Manejo del espacio libre
* Eficiencia y mantenimiento.

**Sistema de manejo de archivos**

* Conjunto de unidades de software que proveen los servicios necesarios para la utilización de archivos: crear, borrar, buscar, copiar, leer, escribir, etc.
* Facilita el acceso a los archivos por parte de las aplicaciones
* Permite la abstracción al programador, en cuanto al acceso de bajo nivel (el programador no desarrolla el soft de administración de archivos)

**Objetivos del SO en cuanto a archivos**

* Cumplir con la gestión de datos
* Cumplir con las solicitudes del usuario
* Minimizar / eliminar la posibilidad de perder o destruir datos
  + Garantizar la integridad del contenido de los archivos
* Dar soporte de E/S a distintos dispositivos
* Brindar un conjunto de interfaces de E/S para tratamiento de archivos
* Proveer soporte de I/O para múltiples usuarios

**Tipos de archivos**

* Archivos regulares
  + Texto plano
  + Binarios
* Directorios

**Atributos de un archivo**

* Nombre, identificador, tipo, localización, tamaño, protección, seguridad y monitoreo.

**Directorios**

* Contiene la información acerca de archivos y directorios que están dentro de él
* El directorio es, en sí mismo, un archivo
* Interviene en la resolución entre el nombre y el archivo mismo
* Operaciones en directorios: buscar un archivo, crear un archivo, eliminar un archivo, listar directorios, renombrar un archivo.
* El uso de directorios ayuda con:
  + La eficiencia: Localización rápida de archivos
  + Uso del mismo nombre de archivo:
    - Diferentes usuarios pueden tener el mismo nombre de archivos
  + Agrupación: Agrupación lógica de archivos pro propiedades/funciones

**Estructura de directorios**

* Los archivos pueden ubicarse siguiendo un path desde el directorio raíz y sus sucesivas referencias
* Distintos archivos pueden tener el mismo nombre pero el full pathname es único.
* El directorio actual se lo llama “directorio de trabajo”
* Dentro del directorio de trabajo, se pueden referenciar los archivos tanto por su PATH absoluto como por su PATH relativo indicando solamente la ruta al archivo desde el directorio de trabajo.

**Identificación absoluta y relativa**

* Absoluta: El nombre incluye todo el camino del archivo
* Relativa: El nombre se calcula relativamente al directorio en el que se esté.

**Compartir archivos**

* En un ambiente multiusuario se necesita que varios usuarios puedan compartir archivos
* Debe ser realizado bajo un esquema de protección
  + Derechos de acceso
  + Manejo de accesos simultáneos.

**Protección**

* El propietario debe ser capaz de controlar:
  + Que se puede hacer
    - Derechos de acceso
  + Quién lo puede hacer

**Derechos de acceso**

* Los directorios también tienen permisos, los cuales pueden permitir el acceso al mismo para que el usuario pueda usar el archivo siempre y cuando tenga permisos.
* Execution: el usuario puede ejecutar
* Reading: el usuario puede leer el archivo
* Appending: el usuario puede agregar datos pero no modificar o borrar el contenido del archivo
* Updating: el usuario puede modificar, borrar y agregar datos. Incluye la creación de archivos, sobreescribirlo y remover datos.
* Changing protection: el usuario puede modificar los derechos de acceso
* Deletion: el usuario puede borrar el archivo.
* Owner (propietario): tiene todos los derechos, puede dar derechos a otros usuarios.

**Ejemplo - Protección en UNIX**

* Derechos de acceso son definidos independientemente para:
  + (u) user - Creador del archivo
  + (g) group - Grupo
  + (o) other - Todos los usuario del sistema
* Derechos de acceso:
  + (r) Read access
  + (w) Write access
  + (x) Execute access
* Ejemplo:



**Metas del sistema de archivos**

* Brindar espacio en disco a los archivos de usuario y del sistema
* Mantener un registro del espacio libre. Cantidad y su ubicación del mismo dentro del disco.

**Conceptos**

* Sector: Unidad de almacenamiento utilizada en los discos rígidos
* Bloque/Cluster: Conjunto de sectores consecutivos
* File System: Define la forma en que los datos son almacenados
* FAT (File Allocation Table): Contiene información sobre en qué lugar están alocados los distintos archivos.

**Preasignación**

* Se necesita saber cuánto espacio va a ocupar el archivo en el momento de su creación
* Se tiende a definir espacios mucho más grandes que lo necesario
* Posibilidad de utilizar sectores contiguos para almacenar los datos de un archivo
* Qué pasa cuando el archivo supera el espacio asignado?

**Asignación dinámica**

* El espacio se solicita a medida que se necesita
* Los bloques de datos pueden quedar de manera no contigua.

**Asignación continúa**

* Conjunto continuo de bloques son utilizados
* Se requiere una pre-asignación
  + Se debe conocer el tamaño del archivo durante su creación
* File Allocation Table (FAT) es simple
  + Solo una entrada que incluye Bloque de inicio y longitud
* El archivo puede ser leído con una única operación
* Puede existir fragmentación externa
  + Compactación
* Problemas de la técnica
  + Encontrar bloques libre continuos en el disco
  + Incremento del tamaño de un archivo

**Asignación encadenada**

* Asignación en base a bloques individuales
* Cada bloque tiene un puntero al próximo bloque del archivo
* File allocation table
  + Única entrada por archivo: Bloque de inicio y tamaño del archivo
* No hay fragmentación externa
* Útil para acceso secuencial
* Los archivos pueden crecer bajo demanda
* No se requiere bloques contiguos

**Asignación indexada**

* Asignación en base a bloques individuales
* No se produce fragmentación externa
* El acceso “random” a un archivo es eficiente
* File allocation table
  + Única entrada con al dirección del bloque de índices (index node / i-node)

**Gestión de espacio libre**

* Control sobre cuales de los bloques de disco están disponibles
* Alternativas
  + Tabla de bits
  + Bloques libres encadenados
  + Indexación

**Espacio libre - Tablas de bits**

* Tabla (vector) con 1 bit por cada bloque de disco
* Cada entrada
  + 0 = bloque libre 1 = bloque en uso
* Ventaja
  + Fácil encontrar un bloque o grupo de bloques libres
* Desventaja
  + Tamaño del vector en memoria

**Espacio libre - bloques encadenados**

* Se tiene un puntero al primer bloque libre
* Cada bloque libre tiene un puntero al siguiente bloque libre
* Ineficiente para la búsqueda de bloques liber → Hay que realizar varias operaciones de E/S para obtener un grupo libre
* Problemas con la pérdida de un enlace
* Difícil encontrar bloques libres consecutivos

**Espacio libre - indexación**

* Variante de “bloques libres encadenados”
* El primer bloque libre contiene las direcciones de N bloques libres
* Las N-1 primeras direcciones son bloques libres
* La N-ésima dirección referencia otro bloque con N direcciones de bloques libres

**Espacio libre - recuento**

* Variante de indexación
* Está estrategia considera las situaciones de que varios bloques contiguos puedan ser solicitados o liberados a la vez (en especial con asignación contigua)
* En lugar de tener N direcciones libres (índice) se tiene:
  + La reacción del primer bloque liber
  + Los N bloques libres contiguos que le siguen. (#bloque, N siguientes bloques libres)

**Unix - Manejo de archivos**

* Tipos de archivos
  + Archivo común
  + Directorio
  + Archivos especiales
  + Named pipes
  + Links
  + Links simbólicos

**Unix - Estructura de volumen**

* Boot block: Código para bootear el S.O
* Superblock: Atributos sobre el File System
* I-NODE Table: Tabla que contiene todos los I-NODO
  + I-NODO: Estructura de control que contiene la información clave de un archivo
* Data Blocks: Bloques de datos de los archivos

**Unix - Información del I-NODO**

* File Mode: 16-bits flag que almacenan el permiso de acceso y ejecución asociados al archivo
* Link count: número de directorios referenciados a este nodo
* Owner ID: Propietario del archivo
* Group ID: grupo propietario asociado con el archivo
* File Size: Tamaño en bytes del archivo
* Last accessed: Fecha del último acceso al archivo
* Last Modified: Fecha de la última modificación del archivo
* Inode Modified: Fecha de la última modificación del inodo

**LINUX - VFS - Virtual File System**

* Interfaz uniforme para el acceso de los procesos a los archivos
* Es una capa de abstracción sobre los distintos File Systems utilizados
* Asume que los archivos son objetos que comparten propiedades más allá del File System que los almacena
* Objetos
  + Superblock object: Representa el file system montado
  + Inode object: Representa un archivo
  + D-entry object: Representa una entrada en un directorio
  + File object: Representa un archivo abierto asociado a un proceso

**Windows - File System Soportados**

* CD-ROM File System
* Universal Disk Format
* File Allocation Table
  + FAT12, FAT16, FAT32
* New Technology File System (NTFS)

**Windows - FAT**

* FAT es un sistema de archivos utilizado originalmente por DOS y windows 9x
* Porqué windows aun soporta FAT file systems?:
  + por compatibilidad con otro SO en sistemas multiboot
  + Para permitir upgrades desde versiones anteriores
  + Para formato de dispositivos como diskettes
* Las distintas versiones de FAT se diferencian por un número que indica la cantidad de bits que se usan para identificar bloques o cluster
* Se utiliza un mapa de bloques del sistema de archivos, llamado FAT
* La FAT tiene tantas entradas como bloques
* La FAT, se duplicado y el directorio raíz se almacenan en los primeros sectores de la partición
* Se utiliza un esquema de ASIGNACIÓN ENCADENADA
* La única diferencia es que el puntero al próximo bloque está en la FAT y no en los bloques
* Bloques libres y dañados tienen códigos especiales

**Windows - NTFS**

* NTFS es el file system nativo de windows
* Usa 64-bits para referenciar clusters
* Ventajas:
  + Tamaños de archivos y discos mayores
  + Mejor performance en discos grandes
  + Nombres de archivos de hasta 255 caracteres
  + Atributos de seguridad

**ENTRADA/SALIDA**

**Variedad en los dispositivos de I/O**

* Legible para el usuario
  + Usados para comunicarse con el usuario (impresoras, terminales: pantalla, teclado, mouse)
* Legible para la máquina
  + Utilizados para comunicarse con los componentes electrónicos
    - Discos, Cintas, Sensores, etc.
* Comunicación
  + Usados para comunicarse con dispositivos remotos
    - Líneas digitales, modems, interfaces de red, etc.

**Problemas que surgen**

* Amplia variedad
  + Manejan diferentes cantidades de datos, velocidades y formatos.
* La gran mayoría de los dispositivos de E/S son más lentos que la CPU y la RAM.

**Comunicación CPU - Controladora**

* Cómo puede la CPU ejecutar comando o enviar/recibir datos de una controladora de un dispositivo?
  + La controladora tiene uno o más registros
* La CPU se comunica con la controladora escribiendo y leyendo en dichos registros.

**Mapeo de E/S y E/S aislada**

* Correspondencia en memoria (Memory mapped)
  + Dispositivos y memoria comparten el espacio de direcc.
  + I/O es como escribir/leer en la memoria
  + No hay instrucciones especiales para I/O
* Isolated I/O (Aislada, uso de puertos de E/S)
  + Espacio separado de direcciones
  + Se necesitan líneas de I/O. Puertos de E/S
  + Instrucciones especiales

**Técnicas de I/O - Programada**

* CPU tiene control directo sobre la I/O
* CPU espera que el componente de I/O complete la operación
* Se desperdician ciclos de CPU

**Polling**

* En la I/O programada, es necesario hacer polling del dispositivo para determinar el estado del mismo
* Ciclo de “Busy-wait” para realizar la I/O
* Puede ser muy costoso si la espera es muy larga.

**Técnicas de I/O - DMA (Direct Memory Access)**

* Un componente de DMA controla el intercambio de datos entre la memoria principal y el dispositivo
* El procesador es interrumpido luego de que el bloque entero fue transferido

**Interfaz de I/O - Metas**

* Es deseable manejar todos los dispositivos de I/O de una manera uniforme, estandarizada
* Ocultar la mayoría de los detalles del dispositivo en las rutinas de nivel más “bajo” para que los procesos vean a los dispositivos, en términos de operaciones comunes como: read, write, open, close, lock, unlock.
* Eficiencia
  + Los dispositivos de I/O pueden resultar extremadamente lentos respecto a la memoria
  + El uso de la multiprogramación permite que los procesos esperen por la finalización de la I/O mientras que otro se ejecuta
  + I/O no puede alcanzar la velocidad de la CPU

**Aspectos de los dispositivos de I/O**

* Dispositivos por bloques (discos):
  + Operaciones: Read, Write, Seek
* Dispositivos por Carácter (keyboards, mouse, serial portes)
  + Operaciones: get, put.
* Formas de acceso
  + secuencial o aleatorio
* Tipo de acceso
  + Acceso compartido: Disco rígido
  + Acceso exclusivo: Impresora
* Tipo de acceso:
  + Read only: CDROM
  + Write only: Pantalla
  + Read/Write: Disco

**Subsistema de I/O - Servicios**

* Planificación
  + Organización de los requerimientos a los dispositivos
  + Ej: Planificación de requerimientos a disco para minimizar movimientos
* Buffering - Almacenamiento de los datos en memoria mientras se transfieren
  + Solucionar problemas de velocidad entre los dispositivos
  + Solucionar problemas de tamaño y/o forma de los datos entre los dispositivos
* Caching - Mantener en memoria copia de los datos de reciente acceso para mejorar performance.
* Spooling - Administrar la cola de requerimientos de un dispositivo
  + Algunos dispositivos de acceso exclusivo, no pueden atender distintos requerimientos al mismo tiempo: Por ej. Impresora
  + Spooling es un mecanismo para coordinar el acceso concurrente al dispositivo
* Reserva de dispositivos: Acceso exclusivo
* Manejo de errores:
  + El SO debe administrar errores ocurridos (lectura de un disco, dipositivo no disponible, errores de escritura)
  + La mayoría retorna un número de error código cuando la I/O falla.
  + Logs de errores
* Formas de realizar I/O
  + Bloqueante: El proceso se suspende hasta que el requerimiento de I/O se completa
    - Fácil de usar y entender
    - No es suficiente bajo algunas necesidades
  + No bloqueante: El requerimiento de I/O retorna en cuanto es posible
    - Ejemplo: interfaz de usuario que recibe inputs desde el teclado/mouse y se muestra en la screen
    - Ejemplo: Aplicación de video que lee frames desde un archivo mientras va mostrando en pantalla.

**Desde el requerimiento de I/O hasta el Hardware**

* Consideremos la lectura sobre un archivo en un disco:
  + Determinar el dispositivo que almacena los datos
    - Traducir el nombre del archivo en la representación del dispositivo
  + Lectura física de los datos en la memoria
  + Marcar los datos como disponibles al proceso que realizó el requerimiento
    - Desbloquearlo
  + Retornar el control al proceso

**Subsistema de I/O - Estructura de datos**

* El kernel mantiene la información de estado de cada dispositivo o componente
  + Archivos abiertos
  + Conexiones de red
  + Etc
* Hay varias estructuras complejas que representan buffers, utilización de la memoria, disco, etc.

**Subsistema de I/O - Drivers**

* Contiene el código dependiente del dispositivo
* Manejan un tipo dispositivo
* Traducen los requerimientos abstractos en los comandos para el dispositivo
  + Escribe sobre los registro del controlador
  + Acceso a la memoria mapeada
  + Encola requerimientos
* Comúnmente las interrupciones de los dispositivos están asociadas a una función de driver.

**Subsistema de I/O - Driver**

* Interfaz entre el SO y el HARD
* Forman parte del espacio de memoria del kernel
  + En general se cargan como módulos
* Los fabricantes de HW implementan el driver en función de una API especificada por el SO
  + open, close, read, write, etc.
* Para agregar nuevo GW solo basta indicar el driver correspondiente sin necesidad de cambios en el kernel

**Driver - Ejemplo en Linux**

* Linux distingue 3 tipos de dispositivos
  + Caracter: I/O programada o por interrupciones
  + Bloque: DMA
  + Red: Ports de comunicaciones
* Los drivers se implementan como módulos
  + Se cargan dinámicamente
* Debe tener al menos estas operaciones:
  + int\_module: para instalarlo
  + clean\_module: para desinstalarlo
* Operaciones que debe contener para I/O
  + open: abre el dispositivo
  + release: cerrar el dispositivo
  + read: leer bytes del dispositivo
  + write: escribir bytes en el dispositivo
  + ioctl: orden de control sobre el dispositivo
* Por convención, los nombres de las operaciones comienzan con el nombre del dispositivo
  + Por ejemplo, para /dev/ptr: ptr\_open.

**Performance**

* I/O es uno de los factores que más afectan a la performance del sistema:
  + Utiliza mucho la CPU para ejecutar los drivers y el código del subsistema de I/O
  + Provoca context switch ante las interrupciones y bloqueos de los procesos
  + Utiliza el bus de memoria en copia de datos

**Mejorar la performance**

* Reducir el número de context switches
* Reducir la cantidad de copias de los datos mientras se pasan del dispositivo a la aplicación
* Reducir la frecuencia de las interrupciones, utilizando:
  + Transferencias de gran cantidad de datos
  + Controladoras más inteligentes
  + Polling, si se minimiza la espera activa
* Utilizar DMA

**BUFFER**

**Disk Cache**

* Buffers en memoria principal para almacenamiento temporario de sectores de disco.
* Contiene una copia de algunos sectores de disco
* Objetivo: minimizar la frecuencia de acceso a disco.
* El kernel asigna un espacio en la memoria durante la inicialización para esta estructura.
* Un buffer tiene dos partes: el header y el lugar donde se almacena el bloque de disco traído a memoria

**El header**

* Identifica por número de dispositivo lógico y nro de bloque.
* Estado
* Punteros
  + 2 punteros para la hash queue
  + 2 punteros para la free list
  + 1 puntero al bloque en memoria

**Estados de los buffer**

* Free (disponible)
* Busy (en uso por algún proceso)
* Se está escribiendo o leyendo del disco
* Delayed Write (DW): buffers modificados en memoria, pero los cambios no han sido reflejados en el bloque original en disco

**Free List**

* Organiza los buffers disponibles para ser utilizados para cargar nuevos bloque de disco
* No necesariamente los buffer estan vacios
* Se ordenan según LRU (último usado)

**Hash Queues**

* Son colas para optimizar la búsqueda de un buffer en particular
* Se organizan según la función de hash usando (dispositivo, #bloque)
* **Estrategia de reemplazo**
* Cuando se necesita un buffer para cargar un nuevo bloque, se elige el que hace más tiempo que no es referenciado.
* Es una lista de bloques, donde el último es el más recientemente usado (LRU)
* Cuando un bloque se referencia o entra en la cache queda al final de la lista
* No se mueven los bloques en la memoria: se asocian punteros

**Funcionamiento del buffer cache**

* Cuando un proceso quiere acceder a un bloque de la cache hay dos alternativas:

1. Se copia al espacio de direcciones de usuario
2. Se trabaja con memoria compartida

* Cuando un proceso quiere acceder a un archivo, utiliza su inodo para localizar los bloques de datos donde se encuentra este.
* El requerimiento llega al buffer cache quien evalúa si puede satisfacer el requerimiento o si debe realizar la E/S.
* Se pueden dar 5 escenarios

1. El kernel encuentra el bloque en la hash queue y el buffer está libre.

* Se remueve ese buffer de la free list. Pasa el buffer a estado busy. El proceso usa el bloque

1. El kernel no encuentra el bloque en la hash queue y utiliza un buffer libre.

* Se toma un buffer de la free list. Siempre se usa el primero. Se lee del disco el bloque deseado en el buffer obtenido. Se ubica en la hash queue correspondiente.

1. Idem 2, pero el bloque libre está marcado como DW

* El keren debe mandar a escribir a disco el e buffer marcado como DW y tomar el siguiente buffer de la free list. Si también está en DW, sigue con el mismo proceso hasta encontrar uno que no esté marcado como DW. Una vez escritos a discos los bloques DW son ubicado al principio de la FreeList

1. El kernel no encuentra el bloque en la hash queue y la free list está vacía.

* El proceso queda bloqueado en espera a que se libere algún buffer. Cuando el proceso despierta se debe verificar nuevamente que el bloque no esté en la hash queue (algún proceso pudo haberlo pedido mientras este dormía)

1. El kernel encuentra el bloque en la hash queue pero está busy.

* El proceso se bloquea a la espera de que el buffer se desbloquee. Se despiertan todos los procesos en espera de algún buffer. El proceso que buscaba el buffer 99 debe buscarlo nuevamente en la hash queue y en la freelist.