- **4.1 Ficheros**
  - **Fichero**:
    - **Almacenamiento**:
      - **Memoria principal:** Volátil, no persistente. Desaparecen al apagar el sistema y son acedidos por el procesador.
      - **Memoria secundaria:** No volátil, persistente. Organizada en bloques de datos, se necesita una abstracción para simplificar las aplicaciones, fichero.
    - Cuando se lee o escribe se opera a nivel de bloque, no de byte. Se cogen bloques completos.
    - **Sistema de ficheros:** Ofrece al usuario una vision logica simplificada del manejo de los dispositivos periféricos en forma de ficheros. Capa de software entre dispositivos y usuarios.
      - Proporciona un mecanismo de abstracción que oculta los detalles relacionados con el almacenamiento y distribución de la información.
      - **Funciones:**
        - Organizacion.
        - Almacenamiento.
        - Recuperacion.
        - Gestión de nombres.
        - Implementación de la semántica de Coutilizacion.
        - Protección.
      - **Funcion principal:** Establece una correspondencia entre los ficheros y los dispositivos lógicos.
      - **Visión del usuario:**
        - **Visión logica:** Ficheros, Directorios , Sistemas de Ficheros y particiones (simula discos a partir de un dispositivo. La tabla de particiones almacena los nombres, bloques de inicio y fin de cada partición).
        - **Visión física:** Bloques o bytes.
      - **Características para el usuario**: Almacenamiento permanentes de infomacion, no desaparece aunque se apague el computador.
        - Información estructurada de forma logica.
        - Nombres lógicos y estructurados.
        - Se acceden a través de llamadas al sistema operativo o de biblioteca de utilidades.
      - **El acceso a los dispositivos es:** Incomodo y no seguro, si se accede no hay restricciones.
  - **Atributos**:
    - **Nombre**: Identificador legible por una persona.
      - Es característico, para que el usuario lo recuerde, ya que el identificador número es muy complicado.
      - **Longitud**: fija en MS DOS o variable en UNIX, Window.
      - **Extensión**: Fija para cada tipo de ficheros.
      - Sensible a tipografía.
    - **Identificador:** Etiqueta unívoca del archivo.
    - **Tipo de fichero:** Formatos de Ficheros.
    - **Ubicación**: Identificador del dispositivo del almacenamiento y posición dentro del fichero.
    - **Tamaño de fichero:** Numero de bytes en el fichero.
    - **Protección:** Control de accesos y de las operaciones sobre el fichero.
    - **Información temporal:** Creacion, acceso, modificación, etc
  - **Los directorios** relacionan nombres lógicos y descriptores internos de ficheros.
  - **Operaciones:**
    - **Creación:** Asignación de espacio inicial y meta datos.
    - **Borrado**: Liberacion de recursos.
    - **Escritura**: Almacena informacion.

        - **Lectura**: Recupera informacion.
- **Vista logica:**
    - **Estrutura del fichero**:
        - **Ninguna**, secuencia de palabras o bytes.
        - **Estructura sencilla de registros**, en líneas de tamaño fijo o variable.
        - **Estructuras complejas**, documentos con formato y fichero de carga reubicable.
    - Cojunto de informacion relacionada que ha sido definida por su creador.
    - Secuencia o tira de bytes.
    - **Métodos de acceso:**
        - **Acceso secuencia**: Basado en el modelo de acceso a datos en una cinta magnética. Operaciones orientadas a bytes, se puede avanzar hacia alante o atrás, pero no dar un salto.
        - **Acceso directo**: Basado en el metodo de acceso a dispositivos de disco. Fichero dividido en registros de longitud fija. Puede dar saltos, utilizando un puntero de posición para evitar tener que especificar la poscion en todas las operaciones.
- **Semántica de compartición**:
    - **Compartición de ficheros:**
        - Varios procesos pueden acceder simultaneamente a un fichero. Ambos pueden leer a la vez, pero no escribir simultáneamente.
        - Es necesario definir una semántica de coherencia.
    - **Opciones**:
        - **Semántica UNIX:** Las escrituras en un archivo son inmediatamente visibles
            - Un archivo abierto tiene asociado un puntero de posición, el puntero puede ser compartido o uno por proceso.
        - **Semántica de sesion:** Las escrituras sobres archivo abierto no son visibles por otros procesos con el archivo abierto. Cuando se cierra un fichero los cambios son visibles. Será visible para los que habrán el fichero después de la modificacion, pero no antes.
        - **Semantica de archivos inmutables**: Un archivo puede ser declarado como compartido. No se puede modificar, no admite modificación de nombre y contenido.
        - **Semántica de versiones**: Las actualizaciones se hacen sobre copias con numero de version. Visibles cuando se consolidan versiones. Sincronización explicita si se requiere actualización inmediata.
    - **Control de acceso:**
        - **Lista de control de acceso:** Lista de usuarios que pueden acceder a un fichero.
            - Si hay diferentes tipos de acceso una lista por tipo de control de acceso.
        - **Permisos**:
            - Versión condensada.
            - Tres tipos de acceso (rwx)
            - Permisos para tres categorías (usuario, grupo, otros)
- **Llamadas al sistema:**
    - **Abrir**: int open(const char * path, int flags, [mode_t mode])
        - Abre o crea un fichero especificado por path.
        - **Flags**: O_RDONLY, O_WRONLY, or O_RDWR. Se pone solo 1
        - **Mode**: O_CREAT, O_APPEND, O_TRUNC, … . Se separan con |.
        - Devuelve un descriptor de fichero (o -1 si error).
    - **Crear un fichero:** int creat(rutaDelFichero, permisos) La ruta con el nombre del fichero.
        - OJO, creat siempre crea un fichero para solo lectura.
    - **Cerrar**: int close(int fildes)
        - Cierra un archivo abierto anteriormente asociado al descriptor fildes (o -1 si error).
        - Cierra, pero no borra el fichero, para ello se usa el unlink.
    - **Leer**: ssize_t read(int fildes, void * buf, size_t nbyte)
        - Intenta leer de un archivo (fildes) tantos bytes como indica nbyte, colocando la información leída a partir de la dirección de memoria buf.
        - Devuelve el número de bytes leídos (que puede ser menor o igual a nbyte). 0 → Fin de fichero (EOF).

- **Escritura**: ssize_t write(int fildes, const void * buf, size_t nbyte)
  - Intenta escribir en un archivo (cuyo descriptor de fichero fildes se obtuvo de abrirlo) tantos bytes como indica nbyte, tomándolos de la dirección de memoria indicada buf.
  - Devuelve en n el número de bytes que realmente se han escrito (que puede ser menor o igual a nbyte). Si retorno = -1 → Error de escritura.
- **Desplazamiento en un fichero:** off_t lseek(int fildes, off_t offset, int whence) l de long
  - Modifica el valor del apuntador del descriptor fildes en el archivo, a la posición explícita en desplazamiento (offset) a partir de la referencia impuesta en origen (whence). Si retorno = -1 → Error de posicionamiento.
  - **whence** indica desde dónde se salta:
    - SEEK_SET → desde el principio del fichero.
    - SEEK_CUR → desde la posición actual.
    - SEEK_END → desde el final del fichero.
  - offset se expresa en un numero positivo o negativo de bytes.
  - En C se puede saltar mas lejos que el final del fichero, pero estaremos fuera, el sistema operativo asume que sabemos lo que hacemos.
- **Crear un enlace a un fichero**: int link(const char *nombre, const char *nuevo);
  - Se crea un enlace al nodo i del fichero indicado, si se ha creado no se borra hasta eliminar todos los hardlink.
  - Se crea entre ficheros de la misma partición.
  - Comparte el i nodo del enlazado, no tiene uno propio
  - Crea un hard link con nombre nuevo al archivo nombre.
  - Incrementa contador de enlaces del fichero nombre.
- **Crear un enlace simbólico:** int symlink(const char *nombre, const char *nombreenlace);
  - Tiene su propio i nodo, no lo comparte, ya que es un fichero que almacena el nombre/ path del fichero al que apunta. Por lo que no suma al contador de enlaces, por lo que si se quiere borrar el fichero este enlace no afecta y se borrará.
  - Crea un enlace simbólico hacia nombre desde nombre enlace.
  - Crea un nuevo archivo nombreenlace que incluye ”nombre” como únicos datos.
- **Borrar un fichero de disco:** int unlink(const char *nombre);
  - Borra el archivo nombre siempre que NO tenga enlaces hard pendientes (contador enlaces = 0) y nadie lo tenga abierto.
  - Si hay enlaces duros (contador enlaces > 0) , se decrementa el contador de enlaces.
  - Si algún proceso lo tiene abierto, se espera a que lo cierren todos.
  - El fichero debe estar cerrado, con close(,), si no lo está tardara en borrarlo

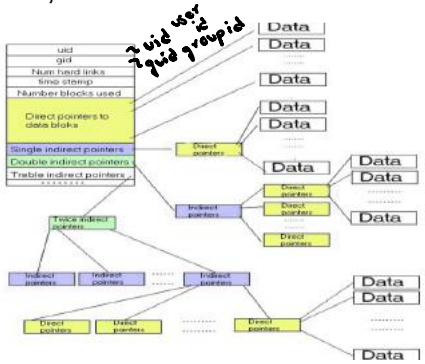
◦ **Representación del fichero**: El sistema debe mantener información sobre el fichero, metadatos, que son dependiente del sistema operativo.

- **Asignación de espacio en disco:** Gestión del espacio libre y ocupado del disco.
  - **Preasignacion**: Asigna en creacion del tamaño maximo posible del fichero, todo el que podría necesitar.
  - **Asignación dinámica:** Asignación de espacio segun se va necesitando, va tomando unidades de asignación segun le hagan falta.
  - **Cuestiones a condisiderar** en cuanto al tamaño de asignación:
    - **Tamaño de asignación grande:** Permite tener toda la información contigua en disco, mayor rendimiento.
    - **Tamaño de asignación pequeño:** Se necesitan mas metadatos, aumenta el tamaño de los metadatos.
    - **Tamaño de asignación fijo**: Reasignación de espacio simple.
    - **Tamaño de asignación fijo y grande**: Incrementa el malgasto de espacio.
    - **Tamaño de asignación variable y grande**: Incrementa el rendimiento, pero aumenta la fragmentación externa.

fragmentación interna →

  - **Fragmentacion**: Para solucionarlo es necesaria la campactacion, elimina los espacios.
    - **Interna**: Cuando queda espacio libre en el ultimo bloque de la secuencia.
    - **Externa**: Cuando hay bloques vacíos que no podemos usar al no haber suficientes contiguos como para introducir informacion.

- **Asignacion encadenada**: Cada bloques contiene un puntero al bloque siguiente, se asignan bloques de uno en uno. No hay fragmentación externa, al poder haber uno suelto mientras apunte a otro. **La consolidación** mejora las prestaciones colocándolos secuencialmente.
  - Se indica la direccion del primer bloque y cuantos bloques hay encadenados.
  - La tabla que lo almacena es FAT.
- **Asignación indexada:** Se mantiene una tabla con los identifacadores de las unidades de asignación que forman el fichero.
  - Puede ser **por bloques**, apunta a los bloques en orden, o **por porciones**, indica el inicio de las secuencias de bloques consecutivos.
- **FAT (File Allocation Table):** Tabla que almacena todos los punteros a bloque enlazados de un fichero, apunta a todos lo bloques. El problema es que para llegar a un bloque lejano debe ir avanzando bloque a bloque, muy costoso. Para que no se pierda si falta un enlace, se almacena en varios sitios.
  - **Representación genérica:**
    - Nombre.
    - Atributos.
    - Fecha de creacion.
    - Nombre del dueño.
    - Puntero FAT, a la tabla de todos lo punteros.
    - Tamaño.
- **UNIX:** Utiliza un nodo i para guardar la información. Permite acceder de una manera mucho más rápida a un determinado bloque gracias a los punteros que actúan como un arbol, ya que se van recorriendo los mas pequeños al principio (directos), pero luego los punteros a bloques de punteros permiten avanzar más rápido. Almacena punteros a bloques de punteros, por lo que comenzando desde el nivel mas alto puede acceder rápidamente a uno específico recorriendo los niveles del arbol. No tiene fragmentación interna.
- **Nodo i:** Almacena la información de un fichero, pero NO almacena ni el nombre ni el puntero de la posición de lectura/escritura (este ultimo, se almacena en la tabla de ficheros abierto
s y puede ser apuntada por varios proceso s o cada uno crea su entrada propia).



  - **FORMA DE ÁRBOL.**
  - Tipo de fichero y protección.
  - Usuario propietario del fichero.
  - Grupo propietario del fichero.
  - Tamaño del fichero.
  - Hora y fecha de creación.
  - Hora y fecha del último acceso.
  - Hora y fecha de la última modificación.
  - **Número de enlaces.**
  - **Punteros directos a bloques (10)**. Guarda los 10 primeros punteros a bloque, de esta manera puede acceder rápido a esos específicos, y si necesita ampliar el almacenamiento se usan los punteros a bloques de punteros.
  - **Puntero indirecto simple**. Puntero a bloque de punteros directos
  - **Puntero indirecto doble**. Puntero a bloque de punteros indirectos simples.
  - **Puntero indirecto triple**. Puntero a bloque de punteros indirectos dobles.
- **Estructura NTFS (Node Tree File System)**:
Es un arbol B+ de bloques, tiene punteros a bloques de punteros y las hojas son punteros directos a bloque. Para que no perder enlaces se hacen copias de NTFS. Además almacena informacion sobre el fichero.