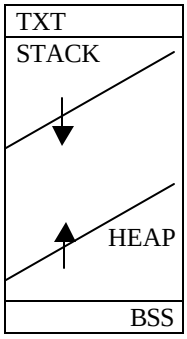
LINK EDICION

1) ¿Qué es el binding? En qué momentos del ciclo de desarrollo de un programa ocurre?

Binding: Es ligar las direcciones simbólicas de un programa para referenciar direcciones a memoria y las propias direcciones de memoria que la máquina interpreta inmediatamente. Puede darse en tiempo de compilación, en tiempo de carga, en tiempo de ejecución (donde la ubicación puede cambiar). Durante la ejecución hay un record de activación que retoca los valores en TXT (área de etiquetas) para poder redirigir el área de datos.

Binding en compilación: vincula las variables estáticas (área BSS).
Binding en ejecución:

- Automático: se retoca el área TXT (llamadas recursivas). Eso es el stack. El SO asigna la dirección inicial del stack para el programa.
- Controlado: la memoria dinámica. El SO debe proveer memoria para que crezca el espacio de variables controladas (heap).



Binding estático: se enlazan las bibliotecas externas precompiladas.
Binding dinámico: enlaza las cosas al vuelo, pero sólo si se usan (especie de dll). El linker pone una llamada a un programita que carga la biblioteca dinámica. Es bueno porque ahorra memoria (pero puede ser una puerta para hackers =). Se da en tiempo de carga y en tiempo de ejecución.

2) ¿Qué tipo de información debe guardarse en los módulos objeto? Describa el uso de cada tipo.

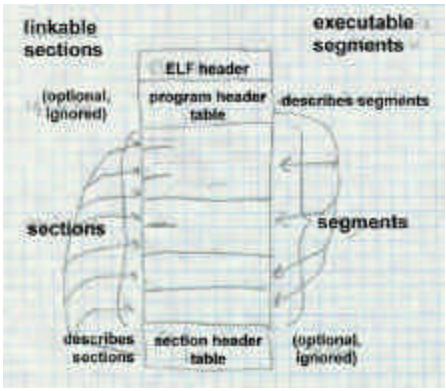
Módulos Objeto:

La estructura de un módulo objeto producido por un traductor es:

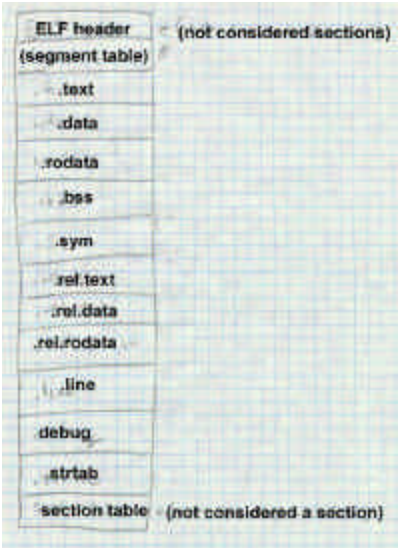
- *Header information:* información general del archivo, como el tamaño del código, nombre del código fuente, fecha de creación.
- *Object code:* instrucciones y datos generados por el compilador o ensamblador.
- *Relocation:* lista de lugares en el código objeto que deben ser fijados cuando el linker cambia las direcciones del código objeto.
- *Symbols:* símbolos globales definidos en este modulo, símbolos que se importan de otros módulos o definidos por el linker.
- *Debugging information:* información adicional utilizada por el debugger. Esto incluye archivo fuente y número de línea de la información, símbolos locales, descripción de las estructuras de datos usadas por el código objeto, como las definiciones de estructuras en C.

3) ¿Qué soluciones se adoptan para guardar esa información en los distintos formatos de archivos objeto y ejecutables como coff, elf y pe?

Objetos ejecutables:
ELF (Executable and Link Format): Se usa mucho en Linux, en los BSD actuales.
Es un formato ejecutable eficiente para memoria virtual con dynamic linking, y hace fácil el mapeo de páginas ejecutables directamente en el espacio de direcciones del programa. Además permite cross-compilation y cross-linking de una plataforma a otra, con la información suficiente en cada archivo ELF para identificar la arquitectura.
El formato ELF se usa para archivos relocatables, ejecutables y objetos compartidos. Archivos relocatables son creados por compiladores y ensambladores pero necesita ser procesado por un linker antes de correr. Archivos ejecutables tienen toda la relocation hecha y los símbolos resueltos excepto, tal vez, algunos símbolos de libraries compartidas que son resueltos en runtime. Los objetos compartidos son librerías compartidas, que contienen información de los símbolos para el linker y código directo para runtime.
Compilers, assemblers, and linkers toman al archivo como un set de secciones lógicas descritas por a section header table; mientras que el system loader lo toma como un set de segmentos descritos por a program header table. Un segmento puede contener varias secciones. Por ejemplo, un segmento de ``loadable read-only'' puede contener secciones de código ejecutable, RO data y símbolos para el dynamic linker. Archivo relocatable tiene section tables, archivos ejecutables tienen program header tables, y los objetos compartidos tienen ambos.



Un archivo relocatable o de objetos compartidos es considerado una colección de secciones definidas en section headers. Cada sección contiene un solo tipo de información como código del programa, RO o RW data, relocation entries, o símbolos. Cada símbolo definido en el módulo es definido relativo a la sección, con lo cual la entrada a un procedimiento va a ser relativo a la sección del código de programa que contiene el código de ese procedimiento.



Un archivo ejecutable tiene el mismo formato que el relocatable, pero los datos están de tal forma que el archivo pueda ser mapeado en memoria y corrido. El archivo contiene un header del programa que sigue al header ELF en el archivo. Ese header del programa define los segmentos que deben ser mapeados.
Objetos compartidos: tiene el program header table en el comienzo, seguido por las secciones en los loadable segments, incluyendo la información para dynamic linking. Las secciones siguientes que abarcan los loadable segments es la tabla de símbolos relocatables y otra información que necesita el linker para crear programas ejecutables que refieren a los objetos compartidos, con la tabla de secciones al final.

PE (Portable executable): Es un formato que se usa en Windows junto con Coff.
Puede contener un directorio de recursos (cursors, icons, bitmaps, menus, and fonts) para todos aquellos recursos que el código usa en ese archivo.
Los archivos ejecutables PE están pensados para la paginación. Las páginas son mapeadas directamente a memoria y corridas. Pueden ser archivos EXE o DLL (dynamic link libraries). Cada uno contiene una lista de funciones que exporta y datos que pueden ser usados por otros archivos PE que estén cargados en el mismo espacio de direcciones; y una lista de las funciones que importa y datos que necesita resolver en tiempo de carga. Cada archivo está formado por un set de divisiones análogos a los segmentos de archivos ELF, que son llamados indistintamente secciones, segmentos u objetos.
Secciones Especiales en archivos PE.

- *Exports*: lista de símbolos definidos en este modulo y que pueden ser vistos por otros. Los archivos EXE típicamente no exportan símbolos, en cambio las DLLs exportan los símbolos para las rutinas y datos que proveen.
- *Imports*: tabla con todos los símbolos de DLLs que necesita resolver en tiempo de carga. El linker predetermina qué símbolos va a encontrar en qué DLLs, con lo cual la tabla comienza con un directorio de entradas a cada DLL referenciada. Cada entrada contiene el nombre de la DLL, identificación del símbolo requerido y el lugar donde va a ser guardado el valor.
- *Resources*: tabla de recursos, organizada como árbol.
- *Thread Local Storage*: Windows soporta múltiples threads de ejecución por proceso. Cada thread puede tener su propio lugar de almacenamiento, Thread Local Storage or TLS. Esta sección apunta a una sección del image usado para inicializar el TLS cuando el thread empieza, y también contiene punteros a rutinas de inicialización para llamarlas cuando cada thread empieza. Generalmente está presente en los EXE pero no en archivo DLL, porque Windows no aloca TLS storage cuando un programa linkea dinámicamente una DLL.
- *Fixups*: If the executable is moved, it is moved as a unit so all fixups have the same value, the difference between the actual load address and the target address. The fixup table, if present, contains an array of fixup blocks, each containing the fixups for one 4K page of the mapped executable. (Executables with no fixup table can only be loaded at the linked target address.)

COFF (Common Obj. File Format): Es un formato que en Unix se usó entre el a.out y elf; y que se usa en Windows.
Un archivo relocatable tiene el mismo header que los archivos PE, pero su estructura es más similar a los archivos ELF.
Cada sección de datos o código lleva información de reubicación y numero de línea. Objetos COFF tienen section-relative relocations, como los archivos ELF, y una tabla de símbolos con aquellos que necesita.
Los archivos COFF provenientes de compiladores de lenguaje no contienen, típicamente, ningún recurso. Por lo general están en un archivo objeto separado creado por un compilador de recursos.
Tiene algunas secciones que no usan los archivos PE. La mas notable es .drective que contiene los comandos de texto para el linker. Los compiladores usan esta sección para informar al linker de buscar las librerías propias del lenguaje. Algunos compiladores que incluyen MSVC también tienen directivas para el linker para exportar código y símbolos al crear una DLL.

Microsoft PE and COFF file
DOS header (PE only)
DOS program stub (PE only)
PE signature (PE only)
COFF header: describe el contenido del archivo (más importante: número de entradas en la tabla).
Optional header (PE only) Contiene punteros a las secciones más usadas.
Section table
Mappable sections (pointed to from section table)
COFF line numbers, symbols, debug info (optional in PE File)

4) ¿Pueden dos sistemas operativos distintos para la misma arquitectura y con el mismo file format de archivo ejecutable ejecutar los mismos programas ejecutables?

No, porque pueden tener diferentes llamadas al sistema.

5) Defina el concepto y el uso de un símbolo relocatable. Dé algunos ejemplos.

Un símbolo es ***relocatable*** cuando la dirección absoluta todavía no está especificada, sino que tiene una dirección que es relativa a una dirección base.
Ejemplo: Una función en un módulo A comienza en la dirección 20. Supongamos que se linkearon otros módulos encima de A tal que la dirección de inicio de A es ahora 50. Entonces la dirección de la función ahora es 70, lo cual implica que todas las referencias a esa función deben ser actualizadas (con el base que es 50).

6) ¿Qué es una referencia externa?

Es una llamada a un símbolo definido en otro módulo objeto, que no coincide con el que realiza la llamada. Como los módulos se compilan por separado, no se puede saber hasta el momento del enlace cuál es la dirección de ese símbolo.

7) ¿Qué es una library, qué símbolos exporta y cuáles importa?

Una library es fundamentalmente una colección de archivos objeto, usualmente con información adicional para hacer más rápida la búsqueda. En otras palabras, es un módulo que contiene subrutinas. Los símbolos que exporta son los puntos de entrada a las subrutinas. Los símbolos que importa son los de las otras libraries que utiliza.

8) ¿Qué es dynamic linking, cómo se diferencian los mecanismos para hacerlo en tiempo de carga y en tiempo de ejecución?

Es la técnica por la cual se enlazan los procedimientos en el momento de ser referenciados (en que se llama) por primera vez. En tiempo de carga, se conoce la dirección del procedimiento porque se está cargando el programa y la biblioteca. En tiempo de ejecución, se pone una dirección inválida en el segmento del enlace y cuando se referencia por primera vez al procedimiento, se carga y se cambia la dirección inválida por la correcta.

9) ¿Cómo se comparten las bibliotecas dll?

El linker busca en las librerías los módulos que resuelven los símbolos externos. Pero en vez de copiarlos, anota en qué library está el módulo, y escribe una lista de libraries en el ejecutable. Cuando se carga el programa, el startup code encuentra esas libraries y las mapea en el espacio de direcciones del programa antes del comienzo del programa.

10) ¿Por qué es necesario y cómo puede implementarse un control de versiones en bibliotecas compartidas?

Es necesario para que cada programa pueda utilizar la versión para la cual fue preparada. Los cambios menores que mantienen compatibilidad conservan el número de versión, pero los cambios mayores, no. Una manera de implementarlo sería proveer un stamp de biblioteca en el archivo ejecutable, entonces se comparan los stamps de los archivos y de las bibliotecas y se elige la correcta.

11) ¿Cómo se cargan las clases en Java?

El runtime environment de java carga, en tiempo de ejecución, las clases que se necesiten. Este componente llamado 'Dinamic class loader' se ocupa de buscar el método solicitado por su nombre con el path de clases del file system.

12) ¿Cómo se mantiene la relación de herencia en memoria para poder hacer un binding en un Lenguaje Orientado a Objetos?

Las relaciones de herencia en los lenguajes orientados a objetos requieren de un binding bidireccional porque un método de una librería puede enviar mensajes a otro del programa que la utiliza. Cuando se invoca un método virtual se exige 'Dynamic Binding', que consiste en hacer búsquedas en la TMV de la clase que invocó al método.

13) Usando la respuesta anterior, explique cómo se realiza un despacho dinámico.

Es el mecanismo que realiza el 'Dynamic Binding'. Este difiere de un lenguaje a otro. En general, son optimizaciones del método virtual, en caso de no ser exitosa la búsqueda en la TMV de la clase que invocó al método, se itera a través de la TMV de los padres hasta encontrarlos.

ADMINISTRACION DE MEMORIA

1) Cómo era el esquema de administración de memoria en particiones fijas? ¿Por qué aparece la protección de memoria?

Monoprogramación: Un solo programa corriendo a la vez compartiendo la memoria con el S.O., de la siguiente forma:

- Usuario tipéa un comando.
- S.O. copia el programa solicitado de disco a memoria y lo ejecuta.
- S.O. imprime el promt y espera otro comando.
- Cuando el usuario ingresa un nuevo comando, el S.O. levanta el nuevo programa a memoria sobrescribiendo el anterior.

Multiprogramación con particiones fijas compartidas: Se divide la memoria en *n* (no iguales) particiones. Estas particiones pueden crearse durante la inicialización del sistema. Cuando un trabajo arriba, éste es puesto en una cola, esperando que una partición que pueda contenerlo se libere. En ese momento el trabajo es levantado a memoria y ejecutado hasta finalizar. Este modelo puede realizarse utilizando una cola de entrada por partición (acolando según el tamaño) o una única cola de entrada para todas. La primera tiene como desventaja que si una cola es muy larga, varios trabajos pueden estar esperando por ser levantados en memoria teniendo gran parte de la misma en desuso. En el segundo caso, es necesario aplicar un criterio para no desfavorecer a los trabajos pequeños y a la vez no desperdiciar particiones grandes en trabajos chicos.

Dado que los programas comparten la memoria, cualquier programa malicioso podría construir una instrucción que le permita saltar a una dirección de memoria fuera de su partición. Algunas posibles soluciones son:

- Clave de protección (en la PSW de IBM360).
- Uso de dos registros especiales, registro base y límite.

2) ¿Cuál es el principio de Swapping?

Consiste en levantar enteramente un proceso a memoria, correrlo durante un determinado tiempo y luego bajarlo nuevamente a disco. Permite variar la cantidad de particiones, su ubicación y tamaño dinámicamente. Cuando un proceso es copiado en memoria, es necesario realocar todas las referencias a memoria. Swapping crea agujeros en la memoria cuando un proceso termina ó crece y debe ser realocado (crece el área de datos o el stack). Estos agujeros pueden ser combinados entre sí para crear otros más grandes. El proceso se denomina compactación y no es usado en la práctica por el alto costo de CPU.

3) Describa algunos métodos de administrar la memoria usando particiones variables.

Bitmaps: Memoria dividida en unidades de alocación, para cada una corresponde un bit en el Bitmap. 0 indica que está libre, 1 ocupado (o viceversa). El tamaño de la unidad es importante. Si es grande se desperdicia espacio en cada unidad. Por el contrario si la unidad es pequeña, el bitmap crece y las búsquedas resultan muy lentas.

Listas enlazadas: Lista doblemente enlazada donde cada entrada corresponde a un segmento, el cual puede ser un proceso o un agujero. Se mantiene ordenada por dirección de memoria (costo de mantenimiento). Algoritmos para alocar un nuevo proceso:

- *First Fit:* El memory manager recorre la lista hasta que encuentra un segmento en el que cabe el proceso. Parte el segmento en dos, ubicando el proceso en uno y dejando un agujero en el otro (en caso que el segmento tenga un tamaño mayor del que el proceso necesita). Este método realiza la búsqueda más corta posible.
- *Next Fit:* Idem first fit, salvo que mantiene un registro de donde terminó la búsqueda anterior, en la siguiente búsqueda comienza desde allí. Tiene peores resultados.

- *Best Fit:* Busca el menor agujero donde quepa el proceso. Es más lento, ya que recorre toda la lista y produce un mayor gasto de memoria, ya que llena la misma de pequeños agujeros donde no cabe ningún proceso.
- *Worst Fit:* El inverso del anterior. No produce buenos resultados..
- *Quick Fit:* Separa en listas, agujeros de algunos de los tamaños mas frecuentemente usados. Es muy rápido pero desperdicia memoria como el best fit.

4) ¿Cómo funciona la Memoria Virtual?.

La idea se basa en que el espacio ocupado por un programa, los datos que maneja y su stack pueden superar el tamaño de la memoria disponible. Por lo tanto el S.O. mantiene en memoria aquellas partes del programa que están siendo actualmente usadas y el resto las guarda en disco. La mayoría de los sistemas con memoria virtual usan una técnica llamada *paging:* existe un set de direcciones de memoria que los programas pueden generar (espacio de direcciones virtuales) usando índices, registro base y otras técnicas. Dichas direcciones son mapeadas a direcciones físicas en la memoria. Desde luego el espacio de direcciones virtuales es significativamente mayor que espacio de direcciones de la memoria. Ambos se dividen en unidades, de igual tamaño, llamadas virtual pages y page frames respectivamente. Para realizar el mapeo entre virtual pages y page frames se utiliza una tabla de páginas o page table. La dirección virtual está formada por el número de página virtual (índice de entrada a la tabla de páginas) y por un offset. El mapeo se lleva a cabo de la siguiente forma:

- Con los primeros *n* bits de la dirección virtual (virtual page) se accede a la correspondiente entrada en la page table.
- Dada la entrada, se verifica si el page frame correspondiente a dicha virtual page se encuentra en memoria (bit presente/ausente).
- De ser así, la dirección física en memoria se obtiene concatenando el número de page frame y el offset.
  En caso de no encontrarse en memoria el page frame (page fault), es necesario leerlo de disco. Previamente debe elegirse otro page frame (que no haya sido usado recientemente), grabarlo en disco (si fue modificado), para escribir el nuevo page frame en ese lugar de memoria.

Lo importante a recalcar es que la memoria virtual (por como está diseñada) hace una separación entre direccionamiento y almacenamiento.

5) Describa el hardware asociado al manejo de las direcciones en memoria virtual.

MMU: Memory Management Unit. Componente del hardware que se encarga de realizar el mapeo de las direcciones virtuales a direcciones físicas en la memoria, verificando si el page frame esta mapeado en memoria y llamando al S.O. en caso de que ocurra un page fault. La MMU puede encontrarse en el CPU o cerca del mismo.
La respuesta 6 puede ser parte de ésta respuesta ya que la TLB es otro mecanismo, o es parte del hardware asociado al manejo de las direcciones en memoria virtual.

6) ¿Para que se usa la memoria asociativa (TLB) en un esquema de memoria virtual?

Se usa para evitar el overhead que se produce por estar obligado a realizar múltiples referencias a memoria por cada dirección virtual de un programa. La solución parte del siguiente fenómeno observado: la mayoría de los programas poseen una gran cantidad de referencias a una pequeña cantidad de páginas. Hay solo un reducido número de páginas que son intensamente leídas. La solución consiste en un pequeño dispositivo de hardware que mapee direcciones virtuales en direcciones físicas en memoria sin utilizar la page table. Translation Lookaside Buffer (TLB). Generalmente se encuentra dentro de la MMU y consiste en un pequeño número de entradas conteniendo la misma información que una entrada en la page table. Como funciona:

- Cuando una dirección virtual se presenta en la MMU, el hardware chequea si el virtual page number se encuentra en la TLB, comparando todas las entradas simultáneamente.
- Si es así, y el acceso no viola los bits de protección, el page frame es extraído directamente de la TLB sin usar la page table.

- Si la página está presente pero hay una violación de permisos, se produce un protección fault tal como hubiera ocurrido con la page table.
- Si la página no se encuentra, la MMU hace una búsqueda ordinaria en la page table, luego sobrescribe alguna entrada de la TLB con la nueva página solicitada.

7) Describa algunos algoritmos de paginación.

Optimal: Imposible de implementar. Numera cada página teniendo en cuenta cuantas instrucciones faltan ejecutarse para que dicha página sea necesaria (no hay forma de saberlo). Al producirse el page fault, remueve de memoria la página cuyo uso es menos próximo. Sirve solo como guía para evaluar otros métodos.

Not Recently Used: Cada página en la page table posee dos bits: R, indicando si la página está siendo referenciada y M cuando la página está siendo modificada. Cuando se inicializa el sistema ambos bits están en cero en todas las entradas de la tabla. En cada interrupción del reloj el bit R es borrado para diferenciar las páginas que fueron recientemente utilizadas. Cuando ocurre un page fault, el sistema inspecciona las entradas clasificándolas según cuatro clases (posibles combinaciones de bits R y M, Clase1: 00, Clase2: 01, Clase3: 10 y Clase4: 11). El algoritmo remueve una página, en forma random, de la clase con menor número.

FIFO: El sistema mantiene una cola con referencias a todas las páginas que actualmente están en memoria. Al producirse un page fault se remueve la página mas vieja (cabeza) y la nueva pagina (página solicitada) se atacha al final (cola). Este método no es aplicable ya que no tiene en cuenta el uso reciente de las páginas, sino el tiempo que llevan en memoria, removiendo páginas que están siendo intensamente usadas.

Second Chance: Idem que FIFO, salvo que inspecciona el bit R, en caso de estar en 1, la página no es removida, sino que es retrasada en la lista hasta el final de la misma. Por lo tanto la búsqueda en la lista continúa hasta encontrar una página no referenciada. De no haber ninguna, se remueve la que inicialmente estaba el comienzo (FIFO).

Clock Page: Mejora para Second Chance: “se usa una lista circular”. Cuando ocurre un page fault, se inspecciona el bit R de la página que está siendo apuntada. Si él es 0, se remueve la página, se ubica la nueva en esa posición y se avanza el puntero a la siguiente página. Si es 1, se avanza el puntero y se inspecciona la siguiente página hasta encontrar una que no esté siendo referenciada.

Least Recently Used: Se basa en mantener una lista enlazada ordenada por el tiempo que transcurrió desde la última vez que una página fue usada. Los resultados del método son muy cercanos al óptimo, sin embargo el mantenimiento de la lista (en cada referencia) es demasiado costoso. Existen algunas posibles implementaciones usando hardware que eliminan el problema.

Working Set: (ver 9).
Conceptos:

- Paginación en demanda: una pagina es alocada en memoria sólo cuando se produce un page fault.
- Pre-paginación: las páginas son alocadas entes de que las mismas sean solicitadas evitando un page fault.

El modelo funciona de la siguiente manera: el sistema mantiene registro del working set de un proceso y se asegura de que el mismo esté completamente en memoria antes de dejar ejecutarse el proceso. Basados en el fenómeno antes mencionado de que un proceso tiene una gran cantidad de referencias a un pequeño set de direcciones, puede decirse que a medida que el proceso de ejecución alanza el working set tiende a mantenerse invariante. Por esto último se consigue reducir la cantidad de page faults. Puede producir *trashing* (ver 10).

WSClock: Idem anterior, además maneja una lista circular para mejorar el rendimiento del método a la hora de buscar una página para remover.

8) Describa alguna forma de mantener el tamaño de las page tables en un valor manejable.

Dado que el espacio de direcciones virtuales puede ser muy grande la page table también resulta serlo. En un modelo en el que la tabla, en su totalidad, se encuentra en memoria, esto se traduce en un gasto inmenso de recursos. Como solución puede implementarse el uso de Multilevel Pages. Consiste en una tabla con varios niveles de indirección (ver gráfico pag.208). La ventaja es que si bien la tabla puede tener millones de entradas, la misma puede usarse teniendo solo un reducido número de ellas en memoria.

9) ¿Qué es el working set?

Es el set de páginas que un proceso está utilizando en ese momento. Si un proceso tiene la totalidad de su working set en memoria, el mismo podrá ejecutarse sin producir ningún page fault, hasta pasar a otra fase de ejecución.

10) ¿Qué es el trashing?

Se produce cuando el espacio libre en memoria es demasiado pequeño, produciéndose un page fault por cada instrucción que contenga referencias. En un modelo de working set, un proceso podría causar un page fault cada vez que el sistema intenta levantarlo.

11) ¿Qué problemas debe tener en cuenta un diseñador de un sistema de paginación?

Alocación Local vs. Global: Determinar si conviene usar algoritmos de reemplazo locales (busca solo en las páginas alocadas por el proceso) o globales (busca en toda la memoria). En general los algoritmos globales funcionan mejor. Los algoritmos locales hacen crecer el working set produciendo trashing. Los algoritmos globales obligan al sistema a decidir cuantos page frames por proceso mantener en memoria (inspeccionando el tamaño del working set). Por lo general el buen funcionamiento de un reemplazo local o global varía según el método de paginación que se use.

Load Control: Cuando el sistema comienza a hacer trashing puede tomarse la decisión de remover algunos procesos de la memoria, en forma temporal (bajarlos a disco) para mejorar el rendimiento de la paginación.

Tamaño de la página: Para determinar el tamaño de la página es necesario balancear ciertos factores que compiten entre sí. Pequeño: Reduce la fragmentación interna (desperdicio). Reduce la cantidad de datos que se mantienen en memoria sin ser usados realmente. Aumenta el tamaño de la page table, generando incluso mas desperdicio de memoria que la fragmentación interna.

Separar instrucciones de datos: Amplía el espacio disponible en la memoria. Cada espacio, el de instrucciones y el de datos maneja su propio page table. Fácil de implementar.

Compartir Páginas: Permite eliminar el desperdicio de tener dos copias de la misma página en forma simultanea en la memoria. Por lo general se comparten las páginas de código.

Política de limpieza: Utilización de procesos en background (demonios) que duermen la mayoría del tiempo. Cuando despiertan inspeccionan el estado de la memoria. Mantienen un pool de page frames no referenciados (cuyas copias en disco fueron actualizadas si fue necesario) que serán removidos por el algoritmo de reemplazo. El page frame removido puede ser obtenido nuevamente del pool de frames.

Interface de VM: la MV no es del todo transparente al programador. Existe la posibilidad de permitirle al mismo manipular el mapeo de memoria. Por ejemplo:

- Permitir que dos o más procesos compartan la memoria.
- Memoria Compartida Distribuida (a través de una red).

12) Describa la memoria segmentada.

Memoria dividida en espacios de direcciones totalmente independientes unos de otros. Cada segmento puede crecer o achicarse sin afectar a los demás. Una dirección se compone de un número de segmento y una dirección dentro del segmento. Los segmentos pueden contener procedimientos, un arreglo, una pila, variables y demás, pero es raro que contenga una mezcla de diferentes tipos. Los segmentos simplifican el manejo de estructuras de datos que crecen y se achican. Permite compartir librerías y aplicar protección a nivel del segmento.

13) ¿Cómo usa el programador la memoria segmentada?

El programador puede decidir qué datos colocar en qué segmentos. Puede aplicar a los datos una protección en forma individual. Puede utilizar los segmentos para compartir librerías o memoria entre procesos. La forma de hacer uso de la memoria segmentada es desde luego mediante un lenguaje de bajo nivel, un lenguaje ensamblador. Por lo tanto, cuando se habla de programador, se refiere a un programador de lenguaje ensamblador o a un compilador de un lenguaje de cuarta generación que hace uso de la segmentación.

14) ¿Cómo puede combinarse segmentado con paginación?

La combinación surge del problema que se plantea cuando un segmento crece lo suficiente como para no caber en memoria. Sistemas que lo aplican:

MULTICS:

- Trata a cada segmento como una MV.
- Posee una tabla de segmentos, con descriptor por cada segmento. Dado que la cantidad de segmentos puede ser muy grande, la tabla en sí es un segmento y está paginada.
- El descriptor indica si el segmento está en la memoria (o parte del segmento). Si es así el descriptor apunta a la dirección de su page table (cada segmento tiene una page table).
- Cada dirección se compone de dos partes: un número de sector y una dirección en el sector. A su vez la dirección en el sector está formada por un número de virtual page y un offset en la página.
- Además usa una TLB.

PENTIUM: Ver directamente el libro.

Lo que hay que destacar, más allá de cómo lo implementa cada sistema, es el hecho de que todos resuelven primero la SEGMENTACIÓN y luego la PAGINACIÓN, en ese orden.

15) ¿Cómo cambian los principios de paginación en el modelo de Objetos?

Cada objeto tiene su propia *page table*, cada uno usa autopaginación. Los objetos paginan sobre si mismos usando su propio algoritmo. Por lo general usan uno standard (*safe paging*).

16) ¿Cómo afecta la multimedia a la Paginación?

Multimedia y Paginación no son compatibles ya que lo primero necesita una velocidad de proceso constante que la Paginación no puede brindarle.

GRAPHICAL USER INTERFACES

1) Describa como traduce el Sistema Operativo la entrada por teclado y por mouse.

Keyboard: Cada vez que se suelta o aprieta una tecla del teclado se manda una interrupción a la CPU, y el hardware provee el numero de esa tecla (que **no** es el código ASCII) depositándolo en el registro de E/S, le corresponde al driver del teclado extraer el carácter tipeado leyendo ese puerto e interpretar la acción a llevar a cabo. Todo lo demás ocurre por software, la mayoría por el driver del teclado. Si por ejemplo si se apretó la tecla A, se le coloca el código de la tecla A que es 30 en el registro E/S, y el driver debe determinar si es una A mayúscula o minúscula (shift + A) o que combinación de teclas se apretaron , esto lo hace porque guarda registro de qué teclas se apretaron y todavía no fueron soltadas.
En las Pentium, hay un microprocesador en el teclado que se comunica a través de un puerto serial especializado con un chip controlador en el parentboard.

Mouse: La mayoría de los mouse poseen un bola que sobresale en la parte inferior cuyo movimiento representa una traslación en la pantalla; cuando la bola gira, al mover el mouse, esta hace fricción sobre dos rodillos o ejes ubicados ortogonalmente (a 90° entre si) uno paralelo al eje X y otro paralelo al eje Y, como resultado estos ejes giran y representan los movimientos en X (este-oeste) y en Y(norte-sur).
Cada vez que el mouse se mueve una distancia mínima en cualquier dirección o se aprieta o suelta un botón del mouse, un mensaje es enviado a la computadora. Esa distancia mínima se llama mickey, es de 0.1 mm generalmente aunque puede ser modificada por software.
El mensaje enviado está compuesto por tres datos Δx, Δy, botones, se manda la distancia recorrida o variación en **x** y en **y** desde el último mensaje y el estado de los botones. El formato del mensaje dependerá del sistema y del numero de botones que posea el mouse (en general ocupa 3 bytes). El doble clic se produce si los dos clic están los suficientemente cerca en espacio y en tiempo, el software lo decide.

2) ¿Qué diferencias hay entre los gráficos vectoriales y los rasterizados?

Los dispositivos para gráficos vectoriales representan el dibujo como puntos en x y en y, por eso pueden dibujar puntos, líneas, figuras geométricas y texto(plotters). Mientras que los dispositivos para gráficos rasterizados (casi todos) representan el área de salida como una grilla rectangular de puntos llamados píxeles, cada uno de los cuales tiene algún valor de la escala de gris o algún color.
Para el despliegue de gráficos rasterizados se utiliza el adaptador grafico, el cual contiene una memoria RAM en donde se almacena la imagen de la pantalla en modo carácter o modo bit.

3) ¿Cómo es el tratamiento del color en los monitores, que relación tiene con la percepción?

Para el despliegue de gráficos rasterizados se utiliza el adaptador grafico, el cual contiene una memoria llamado video RAM (forma parte del espacio de direcciones que usa la CPU) en donde se almacena la imagen de la pantalla en modo carácter o modo bitmap (cada píxel es representado por 1 , 16 o 24 bits). El chip controlador de video es el que saca los caracteres o bits de la video RAM y genera la señal de video usada por el monitor. El monitor genera tres rayos de electrones que impactan sobre la pantalla horizontalmente dibujando líneas sobre él, estos tres rayos corresponden al rojo, al verde y al azul. Este escanéo que se realiza sobre la pantalla se hace hasta 100 veces por segundo. En un instante dado cada píxel esta formado por cierta intensidad distinta e independiente entre sí de esos tres rayos.
Es posible usar una paleta de colores, por ejemplo utilizando 16 bits por píxel para representar el color, de manera que solo hasta 65536 colores puedan ser vistos al mismo tiempo en el monitor. Esos 16 bits pueden ser utilizados para guardar el valor RGB (red, green, blue) con 5 bits para cada uno, o 6 para el verde (ya que es el color que nos parece más fuerte al lado del rojo y el azul, nuestro ojo es más sensible a él por lo que podemos distinguir más matices).
Este es lo que justamente nuestro sentido visual necesita para ver imágenes, ya que el ojo esta compuesto por bastoncillos y conos, los primeros son sensibles a la luminancia y los segundos están divididos en tres clases, los que son sensibles al rojo, los que lo son al verde y los que lo son al azul. Esto permite la visión en colores ya que la combinación de esos colores

primarios genera todos los demás colores. Pero debemos tener en cuenta que por ejemplo no existe la longitud de onda del color púrpura sino que para los humanos este color es solo una sensación, una combinación de longitudes de onda (la verde, roja y azul con cierta intensidad). Otro asunto a mencionar es que los humanos no ven colores o longitudes de onda, sino que distinguimos colores, porque si estuviésemos en una habitación con luz roja, lo que fuese verdaderamente rojo no lo podríamos identificar. Además ciertos colores nos parecen más fuertes que otros, por ejemplo el verde nos parece mas fuerte que el rojo, y este más fuerte que el azul.

4) Describa el modelo WIMP. ¿Cómo se implementa el WIMP en Win32, y en el X-Windows System?

La GUI (graphical user interface) fue inventada por Engelbart y posee cuatro elementos esenciales denotados por los caracteres WIMP: Windows, Icons, Menus y Pointing device. Los Windows son bloques rectangulares de área de pantalla usados para correr programas. Los Icons son símbolos que pueden ser clickeados para que alguna acción tenga lugar. Las Menus son listas de acciones en las que se puede elegir una. Los Ponting device son el mouse, trackball o cualquier otro dispositivo que permite mover un cursor sobre la pantalla para seleccionar ítems.
El software GUI es implementado en el código de nivel de usuario en los sistemas UNIX, mientras que es parte del sistema operativo en las WINDOWS.

En WINDOWS: la ventana es un área rectangular definida por su posición y tamaño dando las coordenadas en píxeles de dos esquinas opuestas, la izquierda superior y la derecha inferior. Una ventana puede tener title bar, menú bar, tool bar, vertical and horizontal scroll bar y puede ser cambiada de lugar, de tamaño o mini y maximizada. Los programas deben ser informados de estos cambios en el tamaño de la ventana, ya que deben redibujar el contenido de su ventana en cualquier momento. Como consecuencia los programas son orientados a eventos.
El dibujado en pantalla de cualquier cosa es controlado por un paquete de procedimientos llamado GDI (graphical device interface), el cual esta diseñado para ser independiente del dispositivo y de la plataforma. Antes que un programa pueda dibujar en una ventana necesita adquirir un device context que es una estructura de datos interna que contiene propiedades de la ventana como el text color, el font, background color, etc. , la mayoría de las llamadas al GDI usan el device context ya sea para setear u obtener propiedades como para dibujar.
La GDI contiene varias llamadas a procedimientos para obtener y liberar el device context, para obtener información sobre ellos, para fijar y obtener los atributos del device context, para manipular los objetos GDI como pens, brushes y fonts y por supuesto llamadas para realmente dibujar en la pantalla. Estas últimas llamadas caen dentro de cuatro categorías, dibujo de líneas y curvas, dibujo de áreas rellenadas, manipulación de bitmaps y despliegue de texto.
Los procedimientos GDI son ejemplos de gráficos vectoriales. Una colección de llamadas a procedimientos GDI puede ser agrupada en un archivo de extensión wmf, este es un archivo metafile ya que describe un gráfico. Las fotos y los videos no son gráficos vectoriales, sino que sus elementos son escaneados sobreponiendo una grilla encima de la imagen, el valor promedio de rojo, verde y azul de cada cuadrado de la grilla son tomados como muestra y representan el valor de un píxel, tales gráficos se llaman bitmaps. El problema con estos gráficos es que no pueden cambiar de tamaño conservando la forma y son problemáticos si se copian imágenes entre distintos dispositivos, por eso surge la estructura DIB (device independent bitmap) con extensión .bmp que soluciona este problema.
Para las fuentes antes se utilizaban los bitmaps, pero esto requería tener un bitmap por cada letra de tamaño diferente. Ahora se usa las fuentes TrueType que no son bitmaps sino siluetas de caracteres, cada carácter es definido por una secuencia de puntos alrededor de su perímetro. Esto permite que sean fácilmente escalables o cambiados de tamaño, solo se tiene que multiplicar cada coordenada de los puntos por el factor de escala.

X-WINDOWS: Hay dos filosofías sobre como debe ser una terminal de red, un punto de vista dice que la terminal debe tener una gran cantidad de poder de procesamiento y memoria para comunicarse a través de la red con protocolos, el otro punto de vista dice que la terminal debe ser extremadamente simple, básicamente debe desplegar píxeles y no hacer muchos cálculos para hacerla barata. X-WINDOWS pertenece a la primer filosofía y la SLIM a la segunda.

Una terminal X es una computadora que corre programas X y se comunica con otras aplicaciones corriendo en computadora remotas. El programa dentro de la terminal X que recoge el input del teclado, el mouse y acepta comandos provenientes de computadoras remotas se llama **X server**. El servidor se halla en la terminal y se conecta con usuarios remotos llamados **X clients** a través de la red enviándoles los eventos de teclado y mouse y aceptando sus comandos para dibujar en pantalla. Por eso el server debe saber qué ventana está activa y dónde está el puntero del mouse para enviar los eventos. Desde el punto de vista del programa, este es un cliente diciéndole al servidor que haga cosas como desplegar figuras o texto por pantalla.
Es posible montar **X Windows System** sobre un UNIX, lo que hace X realmente es definir un protocolo de comunicación entre el **X client** y el **X server**, sin importar que estén en la misma maquina o separados por km, el protocolo y la operación del sistema es el mismo en todos los casos.
X no es un GUI completo solo es un sistema de ventanas, para completarlo otras capas de software se necesitan, la primera se llama **Xlib** que es un conjunto de procedimientos para acceder a la funcionalidad del X. Otra capa por encima de **Xlib** se llama **Intrinsics**, esta capa maneja botones, scroll bars y otros elementos GUI llamados widgets. Para hacer una buena GUI interface se necesita otra capa por encima, la mas popular se llama **Motif** con el cual los programas se comunican.
La administración de ventanas no es parte del X, se encuentra aparte, por lo que un proceso del cliente llamado windows manager (KDE, GNOME) debe hacerse cargo de la creación, borrado y movimiento de las ventanas en la pantalla. Para hacer esto el windows manager manda comandos al X server diciéndole que hacer. El que se encuentre separado permite que corra remotamente.
Este diseño modular con varias capas y múltiples programas lo hacen muy portable entre diferentes UNIX. Mientras que el GDI es muy complicado de mantener porque los sistemas de GUI y de ventanas están mezclados entre si y colocados en el kernel.
Los mensajes por la red por TCP/IP entre el X program y el X server (en diferentes maquinas) pueden ser de cuatro tipos: comandos de dibujo desde el programa hacia el server, respuestas del server a los pedidos del programa, anuncios de eventos (mouse, teclado, etc.) y mensajes de error.

5) ¿Qué facilidades adicionales provee el X-Window System?

Fue respondida en el punto anterior.

6) Describa las diferencias entre los modelos de eventos de win32 y de X-Window. Compare con el modelo observador-observable de Java.

En WINDOWS los programas son orientados a eventos, las acciones del usuario que envuelven el teclado o el mouse son capturadas por el SO y convertidas en mensajes a la aplicación dueña de la ventana activa. Cada programa posee una cola de mensajes a la cual van todos los mensajes relativos a sus ventanas y posee un loop principal que se va fijando en el próximo mensaje y lo procesa llamando al procedimiento apropiado para ese evento o mensaje. En ciertas circunstancias el SO mismo puede pasar por alto la cola de mensajes y ejecutar directamente algún procedimiento del programa. Cada ventana debe estar asociada a un objeto clase que define sus propiedades, tal objeto se llama wndclass, la más importante de sus propiedades se llama lpfnWndProc que es un puntero a la función que maneja los mensajes dirigidos a sus ventana.
Entonces un programa esta compuesto por un main y una función que maneja los eventos. En el main después de definir las propiedades de la ventana se registra el objeto clase para que WINDOWS sepa a que procedimiento llamar para manejar los mensajes, luego se crea la ventana en la pantalla visualmente. Después viene el loop principal para traducir los mensajes recibidos y despacharlos al SO para que este llame al WndProc, la función que maneja los mensajes recibidos. Esta función recibe eventos como la ventana se crea, se destruye, se repinta, etc.

Como WINDOWS, X es un sistema manejado altamente por eventos, el programa puede especificar qué eventos quiere escuchar o quiere que le manden y cuales no, hay una cola de eventos de la cual el programa lee, pero el SO jamás puede llamar a procedimientos dentro del programa por si mismo como ocurre con windows.

Un concepto clave en X es el recurso, que es una estructura de datos que contiene cierta información como una ventana, fuente, colormap (color palettes), pixmap(bitmaps), cursor y graphic context (similares a los device context de windows). Las aplicaciones crean recursos en X server que pueden ser compartidos por múltiples procesos y que son de vida corta.
Un programa X debe conectarse con el X server, reservar memoria para la ventana, decirle al windows manager de la existencia de su ventana para que la conozca y la administre, crear un graphic context, especificar los tipos de eventos que quiere escuchar, desplegar la ventana por pantalla y entrar dentro de un while donde hay un switch para el manejo de los diferentes tipos de eventos.

Modelo JAVA: cada vez que se crea un elemento del GUI como un botón, JPanel, etc. se debe añadir un listener al elemento si uno quiere que escuche los eventos que ocurren y actuar en consecuencia, por supuesto que se puede agrupar elementos con un mismo listener. Esto se codifica y es interpretado por la JVM.

ADMINISTRACION DE ARCHIVOS

1) ¿Cómo influye la estructura de directorios sobre el problema de naming de los archivos?

Primero veamos que es el problema de naming: En un principio, los sistemas tenían una estructura de un solo directorio, el problema que se presentaba era que los archivos eran identificados por su nombre y en máquinas que utilizaban mas de un usuario, podía darse el caso de que dos usuarios pusieran el mismo nombre a sus archivos sobrescribiéndose los datos uno a otro, la solución a este problema fue crear una estructura de directorios, en un principio fue uno para cada usuario de manera que los usuarios podían en ese directorio nombrar a sus archivos como quisieran, sin temor de sobrescribir los de otro usuario, dado que dos archivos con el mismo nombre pero en distintos directorios no se interfieren; de esta forma la estructura de directorios soluciona el problema de naming, brindando distintos contextos a los usuarios, donde estos pueden repetir nombres que están en otros directorios.

2) ¿Qué tipos de archivos reconoce Windows y cuales reconoce UNIX?

Hay dos visiones para esto, según la primera visión (TANENBAUM), en Windows se reconocen dos tipos de archivos:

Directorios: Son archivos de sistema para mantener la estructura del File System.
Archivos regulares: Son los archivos que contienen los datos del usuario, y a su vez se dividen en archivos *ASCII* y *binarios.*

En UNIX, además de los dos tipos que ve Windows, hay dos tipos de archivos mas, estos son:

Character special files: están relacionados a la Entrada/Salida y se usan para modelar dispositivos seriales.
Block special files: Se usan para modelar discos.

El otro punto de vista (dado en clase y en TANENBAUM), refiere a tipos de archivo en cuanto a su estructura, ejemplos de estos son los BMP asociados a un BITMAP, o los .C asociados a un código fuente en lenguaje C. Desde este punto de vista UNIX no maneja tipos y en Windows los tipos se manejan como asociaciones en el registry entre una aplicación y una extensión de esta forma todos los archivo de extensión cpp estarán relacionados a un compilador de C++, pero windows no se ocupa de controlar el formato, son las aplicaciones las que esperan que los archivos tengan un formato específico.
Con esto también se está protegiendo a los archivos de un uso indebido como puede ser deszipear un archivo que no es ZIP.

3) Dé un ejemplo de programas que obliguen al uso de tipos. Indique como se soluciona el soporte de tipos.

Son varios los programas que obligan al uso de tipos, por ejemplo un compilador de C++ solo aceptará archivos que tengan la extensión C o CPP, de esta manera esta manejando los tipos por medio de su extensión, esto es particularmente útil, dado que de acuerdo al tipo de archivo el compilador deberá realizar operaciones diferentes. Otro programa que exige el uso de tipos es el compresor ZIP.
Las formas en que se resuelve el uso de tipos son principalmente dos:
Con las extensiones de los archivos, esto en Windows sirve, pero no en UNIX, donde las extensiones son convenciones para el usuario, pero no significan nada para el sistema operativo.
La otra forma es colocando un número al principio de los archivos llamado *magic number* donde cada número esta relacionado con un tipo de archivo específico, de esta forma las aplicaciones abren el archivo, y leyendo ese número saben si el archivo es del tipo correspondiente.

4) Dé ejemplos de atributos de un archivo.

Todos los archivos tienen un nombre y sus datos. Además de esto, todos los sistemas operativos los asocian con otra información, esta información extra son los atributos del archivo, los atributos de los archivos varían de un sistema operativo a otro, pero algunos de los que se pueden encontrar son los siguientes:

Atributo	*Significado*
Protección	Quien puede acceder y de que forma
Password	Password necesario para acceder a un archivo
Creator	ID del creador
Owner	ID del actual owner
Marca de solo lectura	0 para lectura escritura, 1 para solo lectura
Marca de Oculto	0 para normal, 1 para oculto
Marca de archivo de sistema	0 para archivos normales, 1 para los de sistema
Marca de ASCII/binario	0 para ASCII, 1 para binario
Marca de acceso random	0 para solo acceso secuencial, 1 para acceso random
Marca de temporario	0 para normal, 1 para borrar el archivo al final del proceso
Marca de bloqueo	0 para desbloqueado, distinto de 0 para bloqueados
Longitud de registros	Cantidad de bytes en un registro
Posición de la clave	Offset de la clave dentro de cada registro
Longitud de la clave	Cantidad de bytes en el campo clave
Fecha de creación	Fecha y hora de creación del archivo
Fecha de último acceso	Fecha y hora del último acceso
Fecha de última modificación	Fecha y hora de la última modificación
Tamaño actual	Cantidad de bytes en el archivo
Tamaño máximo	Máxima cantidad de bytes para el archivo

5) Indique que hacen las siguientes operaciones sobre los archivos: *Create, Delete ,Open, Close, Read, Write, Append, Seek, Get Attributes, Set Attributes, Rename.*

Create: El archivo se crea sin datos. El propósito de este system call es avisar que se usará un archivo y setear algunos de sus atributos.
Delete: Cuando el archivo ya no es necesario, debe ser borrado para liberar espacio en disco.
Open: Antes de usar un archivo un proceso debe abrirlo. El propósito de este system call es permitirle al sistema recuperar los atributos del archivo, y la lista de direcciones de disco en memoria principal, para un acceso mas rápido en posteriores llamadas.
Close: Cuando todos los accesos terminan, los atributos y las direcciones de disco ya no son necesarias, entonces se debe cerrar el archivo para liberar el espacio que estos ocupan en las tablas internas. Cerrar el archivo fuerza a que se escriba a disco.
Read: Para leer datos de un archivo. Generalmente se lee desde la posición actual, la llamada debe proveer la cantidad de datos a leer y un buffer donde ponerlos.
Write: Los datos son escritos a un archivo, generalmente en la posición actual, si esta es el final del archivo, el tamaño del mismo crece, si es en el medio, los datos anteriores se sobrescriben.
Append: Es una forma restringida del write que solo permite escribir al final del archivo, muchos sistemas operativos no la proveen, ya que puede hacerse con un seek y un write.
Seek: Para archivos de acceso random es necesaria una forma de decirle de donde debe tomar los datos. Este system call mueve el file pointer a la posición indicada.
Get attributes: Sirve para obtener los atributos de un archivo.
Set attributes: Muchos de los atributos de un archivo son seteables, un ejemplo son los permisos, para poder realizar ese seteo está este system call.
Rename: Frecuentemente un usuario necesita cambiar el nombre de un archivo, para realizar esta operación está este system call.

6) Indique qué hacen las siguientes operaciones sobre directorios: *Create, Delete, Opendir, Closedir, Readir, Rename, Link, Unlink.*

Create: Crea un directorio que solo contiene las entradas . y .. referidas a sí mismo y al padre respectivamente.

Delete: Borra un directorio, solo se pueden borrar directorios que están vacíos. Un directorio se considera vacío si solo tiene las entradas . y .. .
Opendir: Los directorios pueden ser leídos, por ejemplo para listar su contenido, para ello primero deben ser abiertos como los archivos, esta apertura la realiza este system call.
Closedir: Cuando un directorio ha sido leído debe cerrarse, para liberar el espacio, esto se hace con este system call.
Readdir: Este system call, devuelve la próxima entrada en el directorio, formalmente se puede hacer con el read de archivos, pero obliga al usuario a conocer la estructura del directorio, en cambio esta devuelve siempre una entrada del directorio independientemente de la estructura.
Rename: Sirve para cambiarle el nombre a un directorio.
Link: Linkear es una forma de hacer que un archivo aparezca en varios directorios. Este system call especifica un archivo existente y un Path Name, y realiza un link entre ambos. De esta forma el mismo archivo aparece en varios directorios y con nombres distintos.
Unlink: Se da de baja una entrada del directorio, si ese archivo aparecía solo en el directorio, también se da de baja el archivo, caso contrario solo se da de baja la referencia.

7) Para que se usan los buffers.

Los buffers son capas de memoria que se usan como intermediarios entre Virtual File Systems, amortiguando diferencias de velocidad, y en algunos casos como la grabación de CD´s, asegurando una continuidad y uniformidad de transferencia.

8) Describa un ejemplo del uso de buffers.

Un ejemplo de uso de buffers es la Entrada/Salida a través de una red, vamos a ver un par de formas distintas para ver la utilidad y los inconvenientes solucionados por el uso de buffers:

Entrada:

- *Sin uso de buffers*: El proceso hace un system call Read y se bloquea esperando por un caracter, cada caracter que llega produce una interrupción que le entrega el carácter al proceso y lo desbloquea, el proceso guarda el carácter y repite el system call, realiza este loop hasta que recibe el mensaje completo. El problema de este método es que es muy poco eficiente.
- *Con un buffer de n caracteres:* El proceso es parecido al anterior, solo que en este caso, la interrupción va poniendo los caracteres que llegan en el buffer del usuario, y desbloquea el proceso, recién cuando este buffer se llena, de esta manera se logra mejorar un poco la performance; pero tiene el problema de que la página donde está el buffer debe quedar bloqueada en memoria, y si hay muchas páginas en este estado, la performance de la máquina vuelve a decaer.
- *Un buffer de usuario y un buffer del kernel:* La forma de solucionar el caso anterior es poner un buffer del kernell y cargar ahí los datos entrantes, y cuando este se llena, recuperar la página del buffer del usuario y copiar ahí el buffer del kernell, el problema que presenta esta forma es que si mientras se recupera la página del buffer entran nuevos caracteres, no hay donde ponerlos y se perderían.
- *Doble Buffering:* Para resolver el problema del modelo 3, lo que se hace es agregar un buffer mas en el kernell, de esta forma, cuando se busca la página del buffer del proceso de usuario, los nuevos caracteres que entran se guardan en este nuevo buffer, que luego cuando se llene completará el buffer del usuario, mientras que el buffer original le hace de respaldo, y así van intercambiando los roles.

Salida:
El proceso hace un write para dar salida a n caracteres; en este punto el sistema puede tomar 3 decisiones:

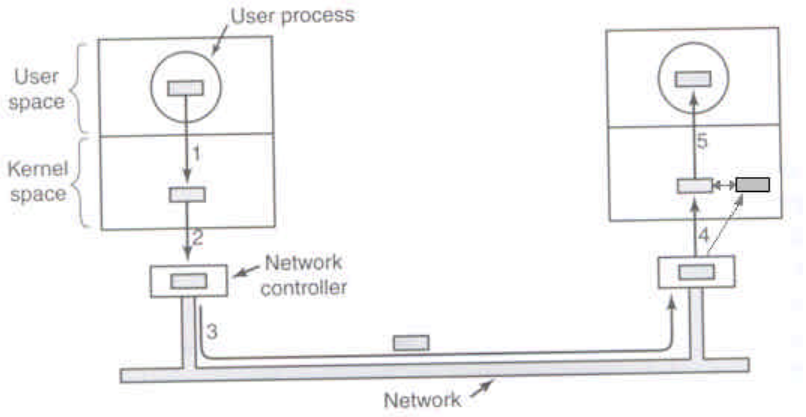
- Bloquear el proceso hasta que todos los caracteres hayan sido transferidos, esto puede ser muy lento si el medio de transmisión es una línea de teléfono
- Puede liberar al proceso para que haga otras cosas mientras termina la transferencia, pero presenta el inconveniente, que para avisarle al proceso que puede reutilizar el buffer hay que realizar interrupciones por software, lo que lleva a una programación complicada y que propicia las Race Conditions.

- Una mejor solución es usar un buffer en el kernell donde se copia el buffer a transmitir, y se libera al proceso, permitiéndole reutilizar el buffer.

Para completar y clarificar hacemos un ejemplo completo de comunicación entre dos máquinas:

1- El proceso hace un write, y el Kernell copia el buffer al buffer del kernell.
2- Cuando se llama al *driver* este copia el buffer del kernell en el buffer del controlador, hace esto para asegura que la transferencia se haga a una velocidad uniforme.
3- El controlador copia el paquete a la red donde la otra terminal va guardando en el buffer del controlador los bytes que llegan
4- El paquete se copia al buffer principal del kernell.
5- Finalmente el buffer se copia en el buffer del proceso receptor.

Generalmente el proceso receptor devuelve una señal de recepción, cuando el emisor del mensaje recibe esta respuesta, puede mandar el siguiente paquete.
Es importante aclarar, que todo este movimiento entre buffers retrasa un poco la transmisión ya que debe hacerse en forma secuencial.



9) Indique las funciones de las siguientes partes del software de I/O: Software de I/O de Nivel del Usuario; Software de I/O independiente del dispositivo; Software dependiente del dispositivo; Software de atención de interrupciones relacionado.

Software de I/O de Nivel del usuario: Tiene como principales funciones a las siguientes:

a) Hacer las llamadas para Entrada/Salida, mediante System calls, los cuales generalmente se hacen en librerías de procedimientos.
b) Dan formato a la Entrada/Salida.
c) Realizan Spooling: Esto es una forma de tratar a los dispositivos de E/S dedicados en un sistema multiusuario, el problema que se presenta es que si el uso de estos dispositivos se deja libre al usuario, puede pasar que un proceso tome control de un dispositivo y lo retenga sin hacer nada, interrumpiendo a los otros procesos; esto se soluciona con este método, que crea un *daemon* y un directorio especial llamado *directorio de spooling* donde se ponen los archivos que requieren hacer uso del dispositivo, y luego es el *daemon* el único que tiene permiso para usar el dispositivo, y este va tomando los archivos en el directorio especial y los procesa en el dispositivo, de esta forma se evita que un proceso acapare un dispositivo.

Software Independiente del dispositivo: Mucho del software de E/S depende del dispositivo, pero hay otra parte que es independiente y es la que implementa esta capa; el borde entre esta capa y los drivers depende del dispositivo y del sistema, dado que hay muchas operaciones que podrían ser independientes, pero que por ciertas causas las implementa el driver. Las funcionalidades que debe cumplir esta capa son las siguientes:

a) Proveer una interfase uniforme para los drivers: Debe proveer una interfase uniforme para cada tipo de dispositivo, de manera que los fabricantes de drivers sepan que funciones deben implementar. Otro aspecto de esta interfase es dar un estándar de cómo se nombraran los dispositivos, por ejemplo en UNIX, un dispositivo de nombre /dev/disk0 apunta a un I-Nodo de un archivo especial a partir del cual se puede acceder al driver específico; con esto de los nombres también se logra proteger a los dispositivos de accesos no permitidos, utilizando el mismo sistema de protección que para los archivos.
b) Buffering: Debe proveer todos los mecanismos de buffering.
c) Reporte de Errores: En la E/S los errores son muy frecuentes, en esta capa se debe dar la estructura del manejo de errores, los errores pueden ser de programación en cuyo caso simplemente se devuelve un código de error al system call, o los errores pueden ser propios del dispositivo y que el driver no pueda resolverlos en cuyo caso la solución puede ser un mensaje preguntando que hacer, o abortar la operación, etc y hay errores sobre estructuras críticas de datos como el directorio raíz, el sistema debe enviar un mensaje de error y abortar.
d) Alocar y liberar dispositivos dedicados: hay dispositivos que solo pueden ser usados de a uno por vez , es función del sistema analizar si los pedidos son aceptados o rechazados dependiendo de si el dispositivo esta disponible o no.
e) Proveer un tamaño de bloque independiente del dispositivo: Los dispositivos trabajan con distintos tamaños de bloque, el trabajo de esta capa es ocultar esta diferencia y proveer un tamaño de bloque uniforme para las capas superiores.

Software dependiente del dispositivo(Drivers): Un driver de dispositivo tiene las siguientes funciones:

a) Aceptar pedidos de lectura y escritura de la capa independiente, y hacer que se lleven a cabo.
b) Inicializar los dispositivos si es necesario
c) Debe manejar los requerimientos de energía del dispositivo.

Todo esto lo hace con conocimiento de la estructura del dispositivo, por ejemplo para lectura o escritura de un disco, el driver debe convertir un número de bloque en los parámetros Cabeza, Cilindro y Sector, y manejar el dispositivo escribiendo y leyendo los registros del controlador del dispositivo.
Los drivers no pueden utilizar system calls, sin embargo en ocasiones necesitan llamar al kernell, para alocar memoria, solicitar un DMA, etc., por ello hay ciertos procesos del kernell a los cuales les es permitido acceder.

Software de atención de interrupciones: Se busca que las interrupciones estén lo más ocultas posibles, para esto lo que se hace es que el driver que lanza la operación de E/S se bloquee a sí mismo.
La función de las interrupciones será desbloquear el driver que disparó la operación de E/S.

10) Describa brevemente como se accede a un dispositivo usando: Modos de I/O programados (PIO), Acceso directo a memoria (DMA), Manejo por Interrupciones.

PIO (Programmed I/O): Con este método la CPU hace todo el trabajo. En este método, la CPU se encarga de hacer la transferencia, quedando esta pendiente de la transferencia que se hace de la siguiente manera:
El acceso consiste en que la CPU entra en un loop en el cual va revisando los registros de los controladores, fijándose cuando está listo para recibir un caracter para mandar a salida, o cuando recibió un caracter de la entrada, y cuando el dispositivo esta listo lee del buffer del controlador, o escribe en el mismo; de esta forma la CPU, esta constantemente revisando los registros del controlador en un estado de *pooling o busy waiting*, no pudiendo hacer otra cosa. Cuando la operación de I/O termina, la CPU devuelve el control al proceso que se estaba ejecutando.
Claramente la desventaja es que ocupa todo el tiempo de CPU hasta que la transferencia termina.
Otra desventaja es que los "modos" PIO indican cuanto tiempo debe transcurrir entre comando y comando de la operación. Esto complica mucho los drivers ya que dicho timing debe ser "calculado" en función de la duración de las instrucciones y esta duración varia a medida que aumenta la velocidad del clock.

DMA (Acceso directo a memoria): Para comprender esto veamos cómo trabaja un control básico de DMA:

a) La CPU setea los registros del controlador de DMA , tales como la dirección base de memoria principal, la cantidad de bytes, en que sentido es la operación (Entrada o Salida). También envía un pedido de lectura o

escritura al control de disco. En escritura, cuando en el buffer del controlador del dispositivo hay datos válidos, comienza el DMA

b) El DMA inicia la transferencia, solicitando una lectura al controlador de disco y pone en el bus de direcciones la dirección donde debe escribirse en memoria. En el caso de escritura, manda el WORD al buffer del controlador de disco.
c) Se realiza la escritura a memoria o a disco
d) El controlador de disco manda una señal de acknowledgement al controlador de DMA. El controlador de memoria aumenta la dirección de memoria y decrementa la cantidad de bytes a leer, y repite los pasos b, y d hasta que la cantidad de bytes a copiar sea 0, en ese momento se emite una interrupción para avisar que se completó la transferencia y desbloquear el proceso que disparó la operación de E/S.

La idea es usar el DMA explicado arriba para realizar la transferencia, en esencia es una transferencia tipo PIO, con la diferencia de que el procesador que se encarga de la transferencia ya no es la CPU, que está haciendo otras cosas, si no el controlador de DMA. Este método es bueno, siempre que el controlador de DMA pueda manejar el bus a alta velocidad y la CPU tiene otros procesos para correr, dado que el controlador de DMA es mas lento que la CPU.

Manejo por interrupciones: La CPU, inicializa el dispositivo con los parámetros necesarios y los buffers; luego de hacer esto la CPU bloquea el proceso que llamo a la E/S y llama al Scheduler para ejecutar otro proceso.
Cuando el dispositivo recibió un nuevo caracter o está listo para sacar otro carácter emite una interrupción. Esta interrupción frena el proceso que estaba corriendo y guarda su estado. Luego el proceso de la interrupción comienza a correr, si no quedan mas caracteres para sacar o recibir, desbloquea el proceso que llamo a la E/S, caso contrario continua con el siguiente byte y retorna el control al proceso que estaba corriendo al momento de la interrupción.
De esta forma la CPU solo se ocupa de la transferencia cuando es necesario.
La desventaja de este método es que tiene muchas interrupciones que desperdician mucho tiempo de CPU.

11) Intente explicar paso a paso como los datos se transfieren desde una aplicación hasta el disco.

Basándonos en las respuestas anteriores el proceso sería de la siguiente forma atravesando por todas las capas de software de I/O:

a) La aplicación en algún momento llama al system call write.
b) Esta llamada será atendida por el software independiente del dispositivo, en esta etapa el buffer a copiar a disco se copiara en un buffer del kernel y se produce una llamada al driver del dispositivo usando la interfase definida en este nivel.
c) El driver del disco toma los parámetros que le pasa la capa independiente y transforma los datos para acomodarlos al dispositivo y completar los registros del controlador de disco (Ej: Transformar a la geometría del disco, la dirección lógica que se le pasa) Cabe destacar, que todas estas transformaciones las puede realizar mediante los datos obtenidos del File System durante una operación de montaje del mismo (En Unix mount). Luego manda mensajes al kernell para solicitar un DMA (en el caso que se use esta transferencia, se pedirán otros servicios en otros casos).
d) Una vez que el driver termina su trabajo, comienza la escritura, durante ese lapso el controlador de DMA le irá proveyendo al controlador de disco los datos a escribir; cuando se termina la transferencia, el DMA emite una interrupción y desbloquea el driver y el proceso que pidió la E/S.