**U0: INTRODUCCIÓN**

**ADMINISTRACIÓN DE MEMORIA**

Programas Procesos Memoria Principal (RAM)

Objetivos:

* Poder ejecutar los programas.
* Poder referenciar sus datos e instrucciones.

Los programas deben ejecutarse en MP, colocando la instancia de ejecución de proceso. De allí el SO puede ejecutar el proceso y referenciar tanto sus instrucciones como datos. El espacio de direcciones disponibles será gestionado por un conjunto de rutinas que constituyen al MMU (unidad de manejo/gestión de memoria).

**Administración en SO Monotareas y Multitareas**

Para *SO Monotareas,* el espacio de direcciones solo será compartido por el SO y por la aplicación o proceso usuario.

En el caso de los *SO Multitareas,* el espacio será compartido por varias aplicaciones usuario.

En ambos casos se utilizan los registros fronteras para delimitar el espacio utilizado por el proceso y para protegerlo del acceso no deseado de otros procesos. En los esquemas multitareas, se requieren registros frontera de inicio y fin para cada proceso.

*Monotareas* *Multitareas*

Usuario

S.O

Usuario 1

S.O

Usuario 2

Apuntadores

Solo se necesita saber el inicio del programa usuario y el tamaño (saco la diferencia para saber dónde debe comenzar el SO)

Se necesita saber el tamaño y los apuntadores de memoria (registros fronteras) para que ningún otro proceso usuario sobrescriba instrucciones del SO.

**JERARQUÍA DE NIVELES DE MEMORIA**

Memoria Secundaria

(Programas)

Registro de Procesador o CPU

(se utiliza para hacer operaciones)

Ejecución (más rápida)

Almacenamiento de Ejecución (RAM)

Almacenamiento de No Ejecución (Discos)

Velocidad

Costo Reg.

Caché

Capacidad

Memoria Principal

(Procesos)

En la composición de la jerarquía de niveles se realiza una distinción entre el MS (caracterizada por contener a los programas) y el resto de los niveles, los cuales son los encargados de almacenar procesos o parte de ellos.

Con la evolución del HW se incorporó una memoria de mayor velocidad conocida como *Caché*, esta memoria permite almacenar partes de procesos o procesos que son utilizados preferencialmente por el SO.

Si bien sumamos un nivel más de transferencia de datos, la pérdida de tiempo que ocasiona la transferencia resulta mucho menor que el tiempo ganado por la ejecución en este nivel superior.

Para que la administración de memoria sea eficiente debe recurrir a un conjunto de estrategias de búsqueda, de colocación y de reposición.

**U1: ADMINISTRACIÓN DE MEMORIA**

Programas **→** Procesos (Código y Datos) **→** Gestionados por el SO **→** MMU (Unidad de Manejo de Memoria) **→** Estrategias

**Tareas de MMU**:

* Definir y asignar el espacio de memoria para ubicar los procesos.
* Interceder para colocar los procesos en ese espacio de memoria.
* Liberar el espacio de un proceso finalizado.

**Estrategias del SO para cada tarea**:

* **De búsqueda**

Estrategias locales y propias para cada proceso. Se utilizan para buscar en el espacio de memoria del proceso la parte del código o datos que el programa necesita.

* + Por demanda: Devuelve el código cuando se solicita.
  + Anticipada: Se anticipa a la solicitud del código, generando un plan de búsqueda anticipado. Es más compleja y requiere más recursos (estructuras auxiliares, caché).
* **De colocación**

Se utilizan para colocar los procesos en las particiones de memoria. Su implementación se aplica en sistemas multiprogramados y al esquema particionado.

Al colocar los procesos se genera una *fragmentación*, que es el espacio de memoria que queda libre y se desperdicia. Puede ser *interna* (se da en particiones fijas y es el saldo de colocar un proceso en un bloque de mayor tamaño que el necesario) o *externa* (es el resto no utilizado de memoria).

Implementaciones:

* + Primer Ajuste: ubica al proceso en la primera partición capaz de albergarlo.
  + Mejor Ajuste: ubica al proceso en la mejor partición que pueda albergarlo dejando el mínimo desperdicio.
  + Peor Ajuste: ubica al proceso en la partición que genere mayor desperdicio. Se utiliza para que esa partición (con el proceso ya alojado) pueda alojar a otro proceso (creando una nueva partición a partir del espacio libre).
* **De Reposición**

Se encargan del intercambio de procesos o partes de los mismos entre MP y MS. Estas técnicas se utilizan para esquemas avanzados de gestión de memoria basados en la operación de ***swapping***, la cual consiste en pasar un bloque de MP a MS. La selección de dicho bloque depende de los algoritmos de reposición:

* **FIFO**

Remueve el bloque que ingresó primero al marco de bloques.

Se comporta mejor con menor cantidad de marcos, yendo en contra de la optimización del uso de los espacios o bloques de memoria, ya que hay menos intercambios. Recordar que éstos suponen pérdida de tiempo y consumo de recursos para gestión.

* **LRU**

Remueve el bloque que no ha sido utilizado o referenciado recientemente.

* **Óptimo**

Consiste en remover el bloque no será referenciado en el futuro o el que será referenciado último (más lejos). En caso de tener varios procesos “víctimas” en igualdad de condiciones (mismo momento de referencia futuro o que ya no serán referenciados) se remueve según FIFO. Es un algoritmo no implementable ya que requiere conocer la totalidad de la secuencia de bloques que cada proceso solicitará. No obstante, este algoritmo resulta útil para comparar la performance del resto de los algoritmos.

* **Reloj**

Se basa en otorgarle una marca de segunda oportunidad a aquellos bloques que fueron referenciados recientemente. Sumado a esto, el algoritmo implementa un puntero que nos indica el bloque a reponer. Este puntero actúa como la aguja de un reloj: se desplazará a la siguiente posición cuando se inserte o reemplace un bloque. Si un bloque es referenciado, el puntero no se mueve y a dicho bloque se le da una marca de segunda oportunidad. Si el puntero apunta a un bloque marcado, se le quita la marca dándole una nueva oportunidad y se mueve el puntero.

**ORGANIZACIÓN O IMPLEMENTACIÓN**

* **Asignación continua/contigua**

Los datos y códigos se ingresan de forma consecutiva, unos detrás de otros. Se coloca un carácter especial al final del proceso para indicar su fin. Es utilizado en SO monoprogramados.

Se utiliza sólo un registro frontera para evitar que los procesos accedan o se almacenen en posiciones no debidas.

Registro Frontera (RF)

* **Particionado**
  + Particiones Fijas:

Memoria dividida en partes iguales por el SO. Cada partición es para un único proceso. Dentro de cada bloque, la asignación es contigua. El particionamiento se da en el arranque del sistema.

Se utiliza un registro frontera y un número que indica el tamaño de las particiones.

SO

P1

P2

P3

P4

P5

P6

P7

P8

RF

**Estrategias para compilar los procesos:**

• Compilación a través de direcciones absolutas: El proceso es compilado para funcionar en una única partición, ya que los datos a ser referenciados están expresados en direcciones rígidas o absolutas. Dichos apuntadores estarán concentrados en una partición específica. Debido a que los procesos solo pueden ser referenciados en una única partición, se generará una cola de procesos para cada partición.

*Ventaja*: Es más simple de implementar y compilar.

*Desventaja*: Se podrían generar colas desbalanceadas que desencadenarían en una ejecución ineficiente de todo el sistema.

SO

A

B

A

B

C

*Se genera desbalanceo ya que “c” podría ir en la segunda cola y agilizar la ejecución.*

• Compilación con direcciones relocalizables: El proceso es compilado utilizando variables o apuntadores de direcciones dinámicos o relativos que dependerán del inicio de la partición más un desplazamiento interno (D) dentro de la partición. El proceso puede ser cargado en cualquier partición, alterando el inicio de la partición.

*Ventaja*: Optimiza el uso de la memoria por no depender de que la partición se encuentre libre.

*Desventaja*: Requiere mayor sobrecarga al momento de la compilación.

Los errores de referencia suelen darse en ejecución.

SO

A

C

B

A

B

C

* + Particiones Variables

La partición se da en función del espacio requerido por cada proceso.

Cuando finaliza un proceso y se libera el espacio del mismo, se genera un espacio libre (fragmentación externa), que puede ser usado luego por otros procesos.

Requiere un registro frontera y un apuntador al comienzo de cada bloque.

RF

SO

P1

P2

P3

P4

P5

Apuntadores de bloque

Qué se puede hacer con las particiones libres:

* Combinación: Pueden combinarse particiones libres adyacentes para formar una sola de mayor tamaño. Su ventaja reside en que consume menos recursos que compactar y no detiene la ejecución.

P1

P2

P3

P4

P5

P6

P1

P2

P3

P7

P6

* Compactación: Consiste en mover los procesos de forma tal de combinar las particiones libres en una única partición al final. Debe detenerse la ejecución de los procesos para realizar la compactación.

P1

P3

P5

P6

P1

P2

P3

P4

P5

P6

Técnica de Recubrimiento

Es una política establecida para los primeros programadores ya que antes la MP era un recurso crítico y muy costoso. Esta técnica o estrategia no es considerada una organización o una implementación de memoria, ya que solo “modifica” al código del programa. Este mecanismo surge como alternativa para prolongar el uso de una memoria escasa ante procesos de mayor tamaño.

Código estructurado de la siguiente manera: Datos Compartidos (D.C) – Entrada (E) – Procesamiento (P) – Salida (S).

Los compiladores del SO cargaban al proceso en partes: Primero el código de entrada (E), el cual era reemplazado por el código de procesamiento (P) para luego ser reemplazado por la salida (S).

Los datos compartidos existían durante la ejecución completa del proceso y contenían las variables de entrada y los valores de salida.

SO

Proceso

D.C

E

P

S

D.C

E

P

S

* **Intercambio**

Es la primera estrategia que aplica la técnica de ***swapping***. Se basa en el esquema de particiones variables y agrega la capacidad de mover un proceso a MS para liberar espacio en MP (se mueven en su totalidad).

Mover los procesos genera pérdidas de tiempo, emplea mayor cantidad de recursos e implica mayor complejidad en el código, pero es preferible antes que rechazar un proceso por falta de memoria: mejora la fluidez. Esta pérdida de tiempo podría ser minimizada si se baja sólo una porción del proceso y no su totalidad.

* **Memoria virtual**

Similar al intercambio, con la diferencia de que aquí la transferencia de MP a MS es al nivel de bloques y no de procesos completos. Los bloques enviados a MS pueden ser del mismo proceso o de diferentes.

Se implementa con dos esquemas básicos y la combinación de los mismos. Surgen así esquemas de páginas, segmentos, paginación segmentada y segmentación paginada.

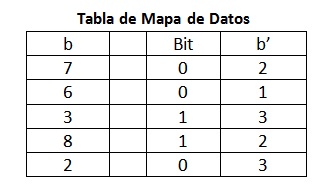
La técnica de MV consiste en generar la ilusión de que se dispone de una memoria de ejecución mayor a la MP real. En este esquema se genera un nuevo espacio de direcciones denominado *“espacio virtual”*. Las direcciones de memoria, dejan de ser direcciones reales para pasar a ser direcciones virtuales, por ende, se requerirá que los procesos sean compilados utilizando direcciones virtuales (el código del proceso va a trabajar con direcciones virtuales, nunca reales). El SO en la MMU deberá implementar mecanismos o esquemas de traducción de direcciones virtuales (ficticias) a direcciones reales (de HW).

* + Dirección Directa

Es el esquema más simple que consiste en generar una tabla de equivalencias entre direcciones virtuales y reales. La MP más la MS es divida en bloques y el tamaño de los mismos podrá ser fijo (*páginas*) o variable (*segmentos*). Los bloques virtuales son identificados sin determinar si se encuentran en MP o MS.

Planteada una dirección virtual, la cual viene definida como el par indicando el *bloque virtual* y el *desplazamiento dentro del bloque*, deberá encontrarse el bloque real correspondiente. Este bloque real sí estará expresado en el esquema de direcciones reales y estará identificado con un *bit de residencia* que indica si se encuentra en MP o MS.

La tabla de traducción o *Tabla de Mapa de Bloques (TMB)* se encuentra almacenada en MP y es gestionada por el SO.

D (A)

SO

1

8

3

5

4

1 2 3 4 5

Virtual Real

6

7

2

1 2 3 *MS*  V = (8,50) (Dirección Virtual)

r = (2,50) (Dirección Real)

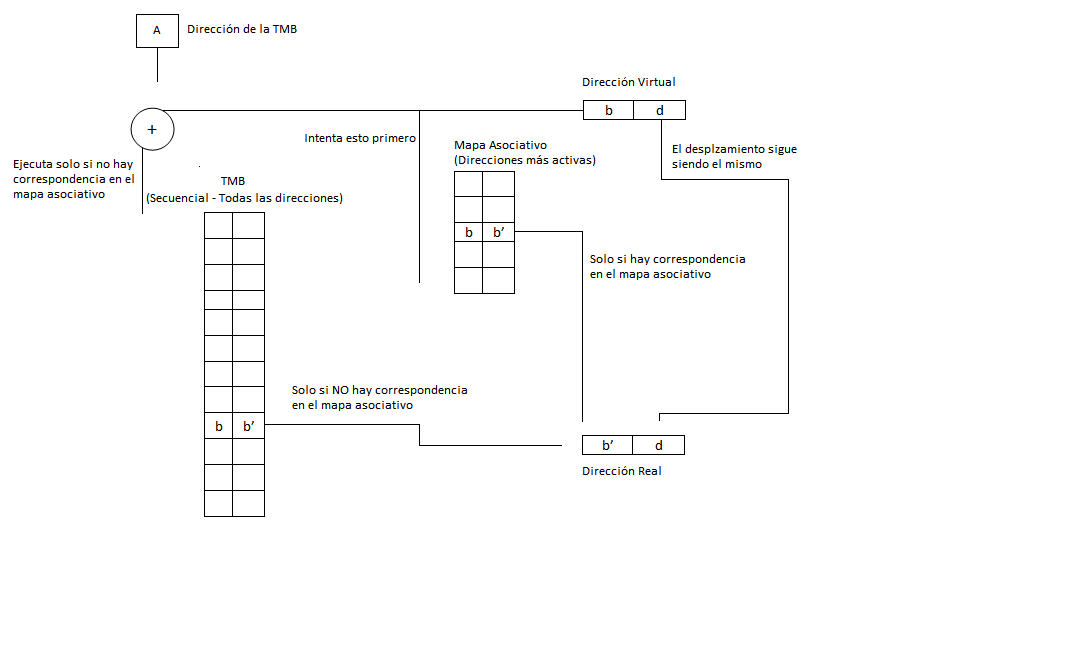
En el gráfico anterior se dispone de un registro apuntador al inicio de la TMB (A), la cual estará compuesta por tantas entradas como bloques tenga la MV. La lectura de la tabla es en forma secuencial, lo cual resulta simple de implementar pero no resulta tener una buena performance o desempeño. La entrada de un bloque determinado estará compuesta por el apuntador al bloque real, el bit de residencia (1 = RAM y 0 = Disco), la longitud en caso de que el esquema lo requiera (sólo segmentos) y un conjunto de bits de acceso o protección que indican qué tipo de operaciones se pueden realizar sobre dicho bloque. Dentro de las operaciones posibles se encuentran: lectura, escritura, ejecución y agregado de datos.

*Ventajas y desventajas*: Relacionado con el número de entradas que posea la tabla. Si los bloques son pequeños, requerirán mayor cantidad de entradas y esto desencadena en un mayor consumo de MP para alojar la tabla de equivalencias. Asimismo, debido a que el acceso o lectura de la misma es secuencial, implicará mayor lentitud en el proceso de encontrar la equivalencia entre bloques. Si el tamaño de los bloques es muy grande, la performance de la traducción mejora pero como contra podemos mencionar el desperdicio o fragmentación dentro del bloque y mayor demora para intercambiar bloques entre MP y MS.

* + Mapa Asociativo / Registro CPU / Memoria Caché

Una forma de acelerar la traducción dinámica de direcciones virtuales consiste en colocar la TMB completa en un *“almacenamiento asociativo”* (Memoria Caché) que tenga un tiempo de ciclo mucho más rápido que la MP. El mapa asociativo posee su propia gestión para contener los bloques que fueron recientemente utilizados alojándolos parciales en la memoria caché. Desventaja: Puede resultar costosa.

* + Traducción Mixta o Híbrida

Implementa la traducción recurriendo a un mapa asociativo, el cual está limitado en la cantidad de entradas pero presenta la ventaja de un rápido acceso directo sobre las mismas. También utiliza la TMB del modelo de traducción directa. La resolución de una traducción de dirección virtual a real comienza por buscar el bloque virtual dentro del mapa asociativo, de no encontrarlo prosigue la búsqueda sobre la tabla de mapa de bloques.

En los esquemas de MV, resulta de gran importancia establecer qué bloques se intercambiarán entre MP y MS. Para dicho fin, el gestor de memoria utiliza las estrategias de reposición basadas en un conjunto de algoritmos. Las estrategias de reposición resultan útiles para elegir la página o segmento que convendrá retirar de la MP en función a los criterios dispuestos por los algoritmos.

Fallos de MV:

* *Fallo de bloque:* Más común y se da cuando un determinado bloque virtual encuentra que su equivalente bloque real no se encuentra en MP. Esto simplemente se puede determinar analizando el valor del bit de residencia.

En caso de que el bit indique que el bloque se encuentra en MS, se genera un fallo que desemboca en una llamada al SO para generar el intercambio de forma tal de que el bloque real se ubique en MP.

* *Fallo de desbordamiento:* Se da cuando el desplazamiento supera la longitud del bloque.

En este fallo también se genera una llamada al sistema para impedir el acceso a una posición que pertenece a otro bloque.

* *Fallo de protección o de acceso:* Se da cuando la operación que se quiere realizar sobre el bloque no está permitida por la configuración de los bits de protección. En este caso, el fallo es capturado y la operación es rechazada, notificando al proceso involucrado.
* *Fallo de posición base:* Se da cuando el bloque indicado no corresponde a un múltiplo (páginas) o a una posición inicial de un bloque (segmento).

En este caso, el fallo es capturado también por el SO, indicando que la dirección de bloque dada no corresponde a la dirección de inicio de un bloque, y se notifica al proceso involucrado.

**U2: SISTEMA DE ARCHIVOS**

Archivos: Son la unidad o entidad mínima de almacenamiento manejada por el SO. Recibe un identificador que caracteriza a dicho conjunto de datos (no es el que se conoce en la interfaz gráfica del SO). Los archivos pueden ser manipulados por el SO con diferentes primitivas o llamadas al sistema tales como open, close, create, copy, delete, etc. Estas primitivas serán traducidas por el SO dependiendo del dispositivo secundario sobre el que se opera.

Sistema de archivos (SA): Es la parte del SO encargada de manipular y gestionar la administración de archivos en forma justa y responsable definiendo un conjunto de funciones para garantizar el acceso controlado a la información de los archivos.

**CONDICIONES DE ALMACENAMIENTO QUE DEBE GARANTIZAR EL SA**

* *Persistencia*. Consiste en que la información no se pierde, es decir, que puede ser referenciada en distintos momentos. Distinguimos entonces información volátil (almacenada en MP) de información persistente (almacenada en MS: discos, CD, DVD, tarjetas de memoria, unidades USB, cintas magnéticas, etc.). La información persistente soporta el no suministro de energía eléctrica.
* *Volumen* de la información. Los archivos permiten almacenar grandes volúmenes de datos, como así también los medios de MS que poseen una *mayor capacidad* que los de MP.
* Posibilidad de *compartir* la información con múltiples procesos, los cuales podrán acceder de manera concurrente sobre los datos almacenados en el SA.

**Organización interna de los archivos**

* *Secuencias de* bytes: Datos almacenados uno detrás de otro, colocando un carácter especial de fin de archivo. El archivo carece de estructura interna y la gestión del SA es muy simple.
* *Registros*: Contienen distintos tipos de datos y generalmente son de una longitud fija definida. A diferencia del esquema anterior, los registros se pueden tratar o acceder de forma independiente.
* *Árboles*: Son un conjunto de registros pero no necesariamente de longitud fija. Cada registro tendrá un identificador o índice por el cual pueda ser accedido en forma directa.

**Tipos de acceso**

Partiendo de la estructura interna del SA, debemos categorizar diferentes tipos de accesos:

* + Lectura secuencial: Leer en forma continua uno a uno los caracteres o registros del archivo. Se aplica para la lectura de bytes y registros fijos. Utiliza la primitiva *read.*
  + Lectura directa: Por medio del índice se accede directamente al registro buscado. Se aplica para árboles. Utiliza las primitivas *seek* y *read*.

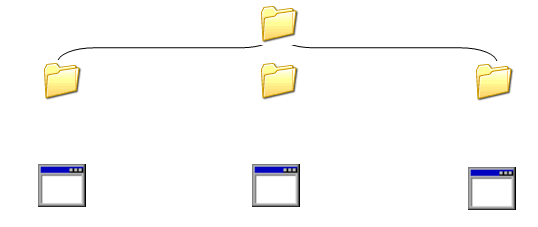
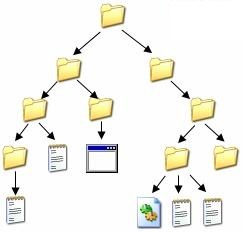
**Funciones del SA**

* Crear, modificar y borra archivos.
* Compartir archivos.
* Respaldar y recuperar archivos.
* Encriptación y desencriptación.
* Independencia de dispositivo.
* Interfaz lógica para usuario final.
* Garantizar la integridad de los datos.

**TIPOS DE SA**

* Archivos regulares: Almacenan datos o información del usuario final y son los que a su vez, se van generando con la intervención diaria del mismo. Estos archivos se pueden manipular en forma libre, directa y sin riesgo alguno para el SO.
* Directorios (Carpetas/Folders): Son archivos para almacenar el mantenimiento de la estructura interna del SA.
* Archivos especiales: son usados por el SO y están relacionados con las operaciones de E/S. Pueden diferenciarse en:
* *Flujo de caracteres:* secuencia de bits que son enviados a los dispositivos de E/S y carecen de estructura interna.
* *Por bloque:* son utilizados por dispositivos destinados para el almacenamiento de datos. Internamente están estructurados en forma de registros y los mismos, pueden ser accedidos en forma independiente.

**Evolución del esquema de Directorios**

****1 2 3

*Nombre relativo*: Nombre del archivo.

*Nombre absoluto*: Nombre que incluye toda la ruta o path del archivo (desde la raíz hasta el nombre del archivo) y lo hace único*.*

En el primer esquema, no es posible encontrar dos archivos con el mismo nombre relativo ya que el nombre absoluto también coincidiría. En cambio, en el segundo y tercer esquema puede que haya archivos con el mismo nombre relativo pero diferente nombre absoluto, ya que se divide en carpetas y luego en usuarios, respectivamente.

**IMPLEMENTACIÓN DEL SA**

Implementación Asignación contigua

Asignación no contigua Lista de sectores

Lista enlazada

Lista con índice Bloque (conjunto de sectores)

Nodo-i

* **Asignación Contigua**

Consiste en colocar los datos del archivo uno detrás del otro desde un inicio a un fin. Al momento de creación, el archivo debe ser dimensionado en el tamaño máximo que se cree que va a alcanzar. Esta implementación utiliza delimitadores (caracteres especiales) conocidos por el SA. Seguramente, al momento de inicializar el archivo, éste ocupará menos espacio real del que realmente requiere, generando desperdicio o fragmentación.

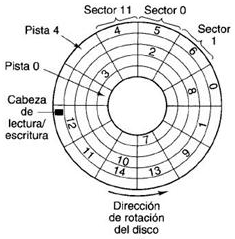
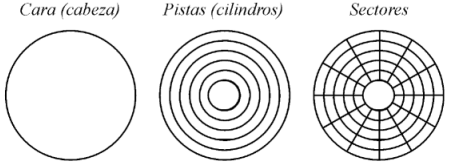
Si el crecimiento del archivo excede el total asignado, será requerido un movimiento total de la información del archivo a una parte del disco que sea capaz de alojar, de manera continua, todos sus datos.

* **Asignación No Contigua**
  + Lista de Sectores

Consiste en colocar el archivo particionándolo en sectores del disco, siendo éstos las mínimas unidades en las que se encuentra dividido el disco. Estos sectores pueden estar dispersos en el disco pero vinculados a través de índices.

*Ventaja*: Fragmentación o desperdicio mínimo (menor al tamaño de un sector) al colocar un archivo en disco.

*Desventaja*: Excesiva cantidad de índices producto de esta subdivisión tan grande. Como esta información de gestión debe ser almacenada, el consumo de espacio resulta realmente significativo.



* + Lista Enlazada

Almacena en el bloque parte de información de gestión (apuntador al siguiente bloque de archivo) y, en el resto, datos.

El directorio posee la información de inicio de cada archivo que contiene. La información de inicio del directorio raíz es conocida por el SO y es a partir de allí que se localizan todos los archivos.

3

800

Nulo

Bloque 500

Bloque 3

Bloque 800

*Desventajas*:

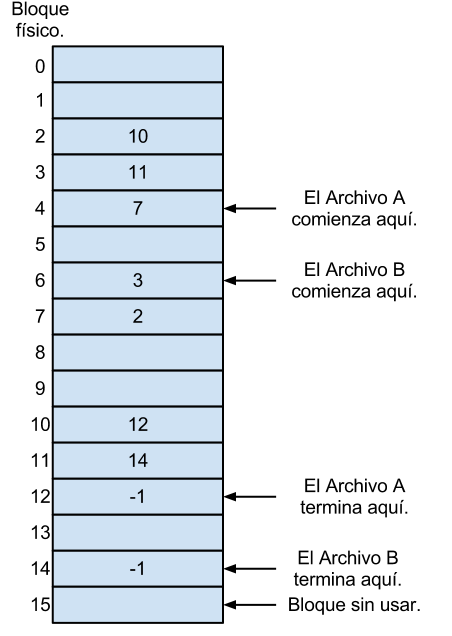
* + - Si bien los índices no consumen MP, su lectura es más lenta.
    - El tamaño del bloque representa menor cantidad de espacio que el bloque usado en MP. Esto quiere decir que si un archivo es “subido” a MP, consumirá menor cantidad de bloques, ya que los mismos son almacenados en forma contigua y no requieren guardar la información de los índices. Suponiendo ahora que un archivo utiliza 5 bloques exactos de memoria, al bajarlo a MS ocupará mayor cantidad de bloques, ya que por cada uno deberá resignar parte de su espacio útil para colocar la información de apuntador al siguiente bloque. El primer bloque bajado ocupa toda la parte de datos del bloque almacenado en disco más una parte del siguiente, y así sucesivamente para el resto de los bloques. Esto desencadena en la utilización de un bloque extra, lo que genera fragmentación.
  + Lista Enlazada con Índice

Subdivide el archivo en bloques, pero sus apuntadores son almacenados en un archivo especial del SO. Este archivo contiene todos los apuntadores de todos los archivos del SA.

En este esquema se posee una *Tabla de Asignación de Archivos* (FAT), que describe la relación entre un archivo y sus bloques en disco.

Esta implementación encadena los bloques utilizando esta tabla, la cual debe ser colocada en su totalidad en MP y se carga al momento cero de iniciar el sistema. No presenta la desventaja del uso de bloques para almacenar información de gestión y datos.

*Desventaja*: En el tamaño de la FAT. Si los bloques son pequeños, la lista es de mayor tamaño, mientras que, si los bloques son grandes, la tabla será de menor tamaño. Además, es importante notar que el sistema no puede grabar continuamente la tabla en disco y una interrupción del suministro eléctrico podría ocasionar escenarios de inconsistencia de archivos (se corrompe FAT y el enlazado posterior se tornaría muy complicado).

La lectura de la tabla puede realizarse comenzando por buscar un bloque que nunca es referenciado, entonces ese es el inicio del archivo.

De todas formas, dicha lectura resulta más sencilla si a partir de un “-1” (final del archivo) realizo el proceso inverso en el recorrido de los bloques hasta llegar a alguno que no sea referenciado.

Archivo A: 4 - 7 - 2 - 10 – 12

Archivo B: 6 – 3 – 11 – 14

* Nodo-i

Se basa en subdividir a los archivos en nodos. Cada nodo posee una estructura interna que consiste en un conjunto de entradas iniciales (que describen los atributos del archivo), un espacio para datos y un conjunto de entradas finales (apuntadores a siguientes nodos).

Atributos

Datos

Apuntadores

El hecho de que cada nodo tenga la información completa del archivo permite cargar los nodos en MP de forma independiente.

Los apuntadores pueden darse en múltiples niveles y generar una estructura jerárquica de nodos, lo que proporciona un acceso más directo que la lectura secuencial.

**CONSISTENCIA DEL SA**

El SO gestiona los bloques ocupados recorriendo desde el directorio raíz todos los apuntadores de los archivos representando los bloques ocupados. Es decir, el SA se basa en los apuntadores de bloques.

En tanto para la gestión de los bloques libres, el SO puede recurrir a dos implementaciones:

* La lista de bloques libres: Contendrá una entrada por cada bloque libre en disco, que estará expresada en binario y la cantidad de bits por entrada será la necesaria para referenciar al mayor bloque .
* Mapa de bits de bloques libres: Utiliza la posición del bit para hacer referencia al bloque en disco. Luego utiliza el 0 y el 1 para indicar si el bloque está ocupado o no (1 = bloque libre).

|  |
| --- |
| **Lista de Bloques Libres** |
| 0000 |
| 1111 |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| **Mapa de Bits de Bloques Libres** |
| 1000000000000001 |

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

*(Blanco = bloque libre)*

El SO implementa una herramienta de análisis de consistencia de archivos. Esta herramienta se encarga de analizar la coherencia del mismo. Como el SA gestiona los bloques libres y los bloques ocupados con diferentes estructuras, se requiere de una herramienta que las compare.

Ésta herramienta genera 2 listas temporales, ambas de capacidad igual a la cantidad total de bloques en disco. Cada entrada de esta lista es un contador de las referencias encontradas al analizar bloques ocupados o bloques libres.

*Ejemplo*:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Bloques Libres | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| Bloques Ocupados | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |

La herramienta obtendrá un caso consistente al obtener 1-0 o 0-1 en las tablas:

|  |  |  |
| --- | --- | --- |
| Bloques Libres | 1 | 0 |
| Bloques Ocupados | 0 | 1 |

Mientras que hallará inconsistencia en los casos:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bloques Libres | 1 | 2 | 0 | 0 |
| Bloques Ocupados | 1 | 0 | 2 | 0 |
| Casos | a | b | c | d |

**Cómo solucionar las inconsistencias**

* **Caso a**
  + *Implica*: Un archivo declaró al bloque como propio ante la herramienta. El 1 en la lista bloques libres implica que se encuentra una entrada del bloque en la lista de bloques libres o que el bit de dicho bloque se encuentra en 1 en el mapa de bits de bloques libres.
  + *Solución*: Eliminar la entrada en la lita de bloques libres o colocar un 0 en la posición del bloque en el mapa de bits de bloques libres.
* **Caso b**
  + *Implica*: La entrada se encuentra duplicada en la lista de bloques libres (este caso sólo puede darse en la implementación de una lista de bloques libres, ya que en un mapa de bits de bloques libres es imposible encontrar entradas duplicadas).
  + *Solución*: Eliminar una de las entradas de la lista.
* **Caso c**
  + *Implica*: Dos archivos se han declarado ser propietarios del mismo bloque.
  + *Solución*: El sistema realiza una copia del bloque doblemente referenciado a un bloque libre y modifica el apuntador de uno de los archivos, reemplazando el apuntador inconsistente al nuevo bloque ocupado. De esta manera, el SA queda consistente, pero uno de los archivos quedará con información inconsistente.
* **Caso d**
  + *Implica*: Ningún archivo lo reclama como propietario, pero tampoco aparece en la lista de bloques libres o tiene un 0 en el mapa de bits de bloques libres.
  + *Solución*: Agregar la referencia en la lista de bloques libres o colocar un 1 en la posición del bloque en el mapa de bits de bloques libres.

Nota: Todas las modificaciones deben hacerse sobre el SA y no sobre la representación de la herramienta, ya que ésta última sólo es temporal y se construye en base a las estructuras manejadas por el SA.

**TIEMPOS QUE MANDAN A LOS DISCOS**

EL SO debe usar las unidades de disco en forma eficiente. Esto implica tener un tiempo de acceso breve y un gran ancho de banda de disco. El ancho de banda del disco es el número de bytes transferidos entre el tiempo total transcurrido desde la primera solicitud hasta la finalización de la última transferencia.

* Tiempo de latencia: Es el tiempo que debe esperar para que el bloque deseado pase por debajo del cabezal de lectura. Se puede mejorar con más revoluciones en el disco.
* Tiempo de transferencia de L/E: Es el tiempo que demanda en mover el contenido del bloque a L/E llevándolo a MP o viceversa.
* Tiempo de búsqueda: Es el tiempo que demora el cabezal de lectura en posicionarse en la pista que contiene al bloque a L/E. Se contabiliza en milisegundos de movimientos del brazo y se soluciona con algoritmos de búsqueda de disco:
* **FIFO** o FCFS: Por orden de llegada. Es intrínsecamente justo, pero generalmente no proporciona el servicio más rápido.
* **SSTF**: Resuelve la petición más cercana al brazo, la que le lleva menos tiempo de movimiento al cabezal. Algunas solicitudes pueden sufrir inanición
* **SCAN**: El movimiento del brazo va y viene del centro al fin del disco. Puede ser ascendente o descendente.
* **C-SCAN**: Igual al SCAN pero solo L/E en un sentido. Da un tiempo de espera más uniforme.
* **LOOK**: Igual al SCAN pero no llega a los vértices del disco, sino que frena en la última petición.
* **C-LOOK**: Igual al C-SCAN pero sin llegar a los vértices.

Con cualquier algoritmo de planificación, el desempeño depende en gran medida del número y tipos de solicitudes.

El método de asignación de archivos puede influir mucho sobre las solicitudes de servicio de disco. Un programa que lee un archivo que asigno contiguamente generará varias solicitudes cercanas entre sí en el disco, y el movimiento del cabezal será limitado. Por otro lado, un archivo enlazado, podría incluir bloques muy dispersos en el disco, y el movimiento del cabezal sería mayor. Por el mismo motivo, la ubicación de directorios y bloques también es importante.

SCAN y C-SCAN funcionan mejor en sistemas en los que la carga sobre el disco es intensa, y no son propensos al problema de inanición.

**U3: SISTEMA DE ENTRADA/SALIDA**

Rutinas del SO para controlar o gestionar la comunicación entre dispositivos.

**Objetivos del SO**

* Optimizar las operaciones de E/S presentando una capa de servicios optimizada para cualquier tipo de dispositivo.
* Facilitar la interacción o el manejo de los dispositivos periféricos.
* Permitir la conexión de cualquier dispositivo sin necesidad de remodelar el SO.
* Conectar y utilizar cualquier dispositivo (Tecnología *Plug and Play*).

**FUNCIONES DEL SO**

* Enviar comandos a los dispositivos: Envía a un dispositivo a utilizar un flujo de caracteres que tiene la operación a realizar y los datos necesarios para realizarla. Además envía el espacio de direcciones donde se van a almacenar los datos propios de la operación y los resultados de la operación.
* Detectar interrupciones: Los dispositivos se van a manifestar hacia el SO por medio de interrupciones. Cada dispositivo tiene un id de interrupción denominado IRQ, el cual permite que el SO pueda saber dónde se produjo el error o interrupción (puede ser un evento, no necesariamente es un suceso negativo).
* Controlar errores: Capturar interrupciones y trabajarlas. Ejemplo: bloques defectuosos de disco, falta de papel en una impresora, etc.
* Proporcionar una interfaz entre los dispositivos y el resto del sistema: Dicha interfaz debe ser lo más *simple* posible y preferentemente *uniforme* para todos los dispositivos. Si el SO tendría que codificarse para cada tipo de dispositivo, éste sería enorme. Por este motivo se diseña una capa genérica por encima del Kernel y por encima de esta capa se implementan todos los drivers o controladores.

**COMUNICACIÓN**

La comunicación real se da entre el SO y la *controladora del dispositivo*. Ésta es la parte electrónica del dispositivo que traduce el flujo de datos enviados por el SO a flujo de datos entendibles por el dispositivo en cuestión. Puede ser parte del dispositivo o ser externa. Una controladora puede manejar uno o múltiples dispositivos.

SO

Controladora

Dispositivo

**CLASIFICACIÓN DE DISPOSITIVOS**

Por flujo de caracteres: Carecen de una estructura y son enviados al dispositivo delimitándolo con caracteres. Manejan archivos especiales por flujo de caracteres.

Por bloques: Los datos que recepcionan y entregan los dispositivos son estructurados y por lo tanto, puede ser identificados y referenciados. Manejan archivos especiales por bloque.

**IMPLEMENTACIÓN O MODOS DE OPERACIONES DE E/S**

* **E/S programada**: Consiste en que el procesador envía la operación de E/S a realizar y el conjunto de datos hacia la controladora. Mientras la operación se realiza, el procesador permanece monitoreando y esperando dicha operación, preguntándole al dispositivo si terminó, si necesita más datos, etc.

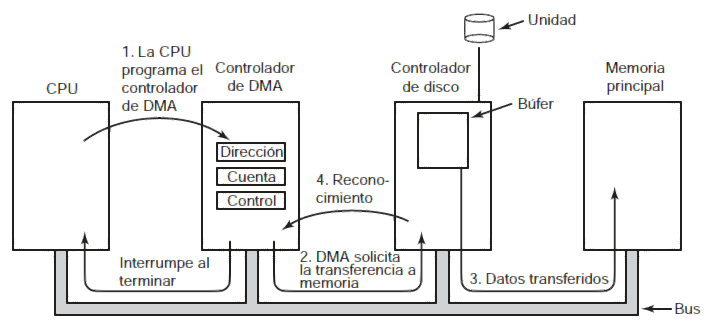
*Desventaja*: Se mantiene al procesador activo para controlar todo el ciclo de vida de la operación de E/S, lo cual desencadena en problemas de performance porque se utiliza el recurso más crítico de la PC en esa única tarea.

* **E/S por interrupciones**: El procesador envía la operación de E/S entregándole a la controladora la información necesaria. Una vez entregada esta información, el procesador cambia de proceso, dejando al proceso inicial en estado bloqueado y continúa con la ejecución del nuevo proceso. En caso de que la controladora quiera notificar algún cambio, debe generar una notificación o interrupción, la cual será capturada por el SO a fin de que el procesador pueda entenderla.

Si bien se mejora el desempeño con respecto a la E/S programada, sigue habiendo bajas de performance dado que el procesador puede ser interrumpido.

* **DMA** (Acceso Directo a Memoria): Surge para resolver las interrupciones que se dan cuando la controladora requiere mayor suministro de datos. El DMA es un chip que actúa como pseudo-procesador. Ante una operación de E/S el procesador entrega los datos al DMA y éste establece conexión con la controladora en cuestión. El procesador continúa con la ejecución de otros procesos. Cuando el DMA recoge todas las respuestas de la controladora (éstas respuestas no las puede procesar), interrumpe al procesador para que las trate. El DMA se encargará de todas las operaciones de movimientos de datos con la controladora para poder concluir con la operación, pero no podrá ejecutar controles que son propios del procesador.

**Operación de Transferencia DMA**



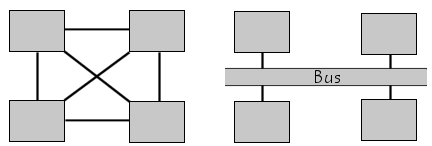
Primero, la CPU programa el controlador de DMA, para lo cual establece sus registros de manera que sepa qué debe transferir y a dónde (*PASO 1*). También emite un comando al controlador de disco para indicarle que debe leer datos del disco en su búfer interno y verificar la suma de comprobación. Cuando hay datos válidos en el búfer del controlador de disco, puede empezar el DMA.

El controlador de DMA inicia la transferencia enviando una petición de lectura al controlador de disco mediante el bus (*PASO 2*). Esta petición de lectura se ve como cualquier otra petición de lectura, por lo que el controlador de disco no sabe ni le importa si vino de la CPU o de un controlador de DMA. Por lo general, la dirección de memoria en la que se va a escribir está en las líneas de dirección del bus, por lo que cuando el controlador de disco obtiene la siguiente palabra de su búfer interno, sabe dónde escribir. La escritura en memoria es otro ciclo de bus estándar (*PASO 3*).

Cuando se completa la escritura, el controlador de disco envía una señal de reconocimiento al controlador de DMA, también a través del bus (*PASO 4*). El controlador de DMA incrementa a continuación la dirección de memoria a utilizar y disminuye la cuenta de bytes. Si la cuenta de bytes es aún mayor que 0, se repiten los pasos del 2 al 4 hasta que la cuenta llega a 0. En ese momento, el controlador de DMA interrumpe la CPU para hacerle saber que la transferencia está completa. Cuando el SO se inicia, no tiene que copiar el bloque de disco en la memoria; ya se encuentra ahí.

*Conceptos*

* Bus de comunicación: Conjunto de conexiones físicas (cables, placa de circuito impreso, etc.) que pueden compartirse con múltiples componentes de HW para que se comuniquen entre sí. El propósito de los buses es reducir el número de rutas necesarias para la comunicación entre los distintos componentes, al realizar las comunicaciones a través de un solo *canal de datos*.



Tipos de buses:

* Bus de datos: Transporta los datos que se transfieren entre unidades. Suele ser bidireccional. Puede ser:
  + *Bus de datos interno*: Transfiere datos entre los elementos de la CPU.
  + *Bus de datos externo*: Comunica la CPU con los periféricos.
* Bus de direcciones: Transporta direcciones de las celdas de la memoria RAM. El tamaño de este bus define la cantidad de memoria RAM que la CPU puede gestionar.
* Bus de control: transporta señales de control y estado (señales de dirección, temporización, interrupción).
* Canal de comunicación: Es el medio de transmisión por que el que viajan las señales portadoras de información.

**SW DE SO PARA E/S**

Los objetivos se logran de una forma comprensible y eficiente al estructurar el SW de E/S en cuatro capas:

1. **Manejadores de interrupciones**:

La idea principal consiste en ocultar las interrupciones en las entrañas del SO.

Para lograr esto, cada vez que un proceso inicie una operación de E/S se bloqueará hasta que termine la operación y ocurra la interrupción. Se bloquea mediante un DOWN en un semáforo, mediante un WAIT en una variable de condición o un RECEIVE en un mensaje. Cuando ocurre la interrupción, el procedimiento de interrupción no realiza lo debido para eliminar el bloqueo que lo inició.

1. **Manejadores de dispositivos** (Driver):

Cada uno de estos controla un solo tipo de dispositivo, a lo sumo una clase de dispositivos cercanos entre sí. Se encargan de cargar comandos en los registros de las controladoras y verificar su ejecución adecuada. Ejemplo:

Manejador de disco: Es la única parte del SO que conoce el número de registros de un controlador de disco y el uso que tienen. Se encarga de los sectores, pistas, cilindros, cabezas, movimiento del brazo, factores de separación, control del motor, etc.

Formalmente, acepta las solicitudes abstractas que le hace el SW independiente del dispositivo y verifica la ejecución de dicha solicitud.

Esa traducción de solicitudes abstractas consiste en decidir cuáles son las operaciones necesarias del controlador, los datos necesarios para las mismas y el orden de ejecución.

Los controladores pueden manejar un solo comando a la vez o pueden aceptar una lista ligada de comandos que pueden llevar a cabo sin ayuda del SO. Entonces pueden ocurrir dos situaciones:

* El manejador tiene que esperar hasta que el controlador realice el trabajo, entonces se bloquea hasta una interrupción.
* La operación requiere microsegundos y termina sin retrasos, entonces no necesita bloqueo.

Independientemente de la situación, después verifica la no existencia de errores y regresa información de estado a quien lo llamó. Si existen otras solicitudes debe seleccionar cuál es la próxima. Sino, se bloquea hasta una próxima solicitud.

Definición:

SW formado por un conjunto de rutinas y tablas que, formando parte del núcleo del SO, ejecutan y controlan todas las operaciones de E/S sobre cualquier periférico, siendo particulares para cada dispositivo.

Contiene tablas con información específica que caracteriza cada tipo y una serie de rutinas que controlan toda la gestión de los mismos y las informaciones que influyen en un sentido o en otro. Se encuentran en MP.

Funciones del driver:

* Define las características del periférico al resto del SO.
* Inicializa los registros asociados al periférico en el momento del arranque (bootstrap) del SO. (Inicialización)
* Habilita y deshabilita el dispositivo para un proceso.
* Procesa todas las operaciones de E/S solicitadas por un proceso. (Atención de peticiones de E/S)
* Cancela toda operación de E/S en el momento que sea necesario por cualquier motivo. (Cancelación de operaciones de E/S)
* Procesa todas las interrupciones de HW generadas por el propio periférico. (Gestión de interrupciones – Interrupt handler)
* Trata los errores y estado del dispositivo haciendo la correspondiente comunicación al usuario. (Time-out controla el tiempo de proceso de la operación y Power-fail actúa en el arranque y al reanudarse el proceso después de un corte de la alimentación.)

Estructura de datos del driver:

* BLOQUE DE CONTROL DEL DRIVER (BCD)

Contiene aquellos parámetros que son susceptibles a ser variados dinámicamente y aquellos que definen el tipo de dispositivo que puede ser atendido por el driver.

* BLOQUE DE CONTROL DE LA UNIDAD (BCU)

Cada dispositivo físico se relaciona desde el punto de vista del SO como una unidad dentro del tipo que corresponda y es definido e identificado por el SO por medio del BCU.

* PAQUETE DE PETICION DE ENTRADA/SALIDA (PES)

Cuando un proceso intenta hacer una operación de E/S, el SO crea un paquete asociado a dicho proceso y a dicha petición para ser tratado por el driver. El paquete se coloca en la cola correspondiente al driver.

1. **SW de E/S independiente del dispositivo**

Función Básica: llevar a cabo las funciones de E/S comunes a todos los dispositivos y proporcionar una interfaz uniforme del SW a nivel de usuario.

Funciones:

* Interfaz uniforme para los manejadores de dispositivos.
* Nombres de los dispositivos: asocia nombres simbólicos de los dispositivos con el nombre adecuado.
* Protección del dispositivo: se encarga de administrar los permisos adecuados por cada tipo de dispositivo.
* Proporcionar un tamaño de bloque independiente del dispositivo.
* Uso de buffers: independientemente que sea dispositivo de bloque o de carácter.
* Asignación de espacio en los dispositivos por bloques.
* Asignación y liberación de los dispositivos de uso exclusivo.
* Informe de errores. Los errores son tratados por el manejador.

1. **SW de E/S en el espacio del usuario**

Aunque la mayoría del software de E/S está dentro del SO, una pequeña parte de él consta de bibliotecas ligadas entre sí con los programas del usuario e incluso programas completos que se ejecutan dentro del núcleo.

Las llamadas al sistema generalmente son llevadas a cabo por procedimientos de biblioteca. El procedimiento de biblioteca será ligado con el programa y contenido en el programa en binario presente en la memoria en tiempo de ejecución.

No todo el SW de E/S a nivel usuario consta de procedimientos de biblioteca. Otra categoría es el spooling, que es una forma de trabajar con dispositivos de E/S de uso exclusivo en un sistema de multiprogramación. Consiste en crear un proceso especial, llamado un demonio y un directorio especial, llamado directorio de spooling. El demonio es el único con permiso para usar el archivo que se crea en el directorio especial.

Direcciones de E/S del dispositivo:

Los dispositivos o sus controladores constan de registros que pueden ser leídos o escritos por el SO para facilitar operaciones que se efectúan sobre ellos. Ellos son:

* Estado: el SO solo puede leer sobre él. (Interrupciones habilitadas o no, error de operación, configuración).
* Operación: el SO a través del driver puede escribir indicando que operación es la que se solicita
* Datos: donde se depositan los datos relativos a la operación.

**U4: SEGURIDAD Y PROTECCIÓN**

**SEGURIDAD EN LOS SO**

Uno de los mayores problemas que se presenta al almacenar información en un computador es la seguridad de la misma, teniéndose que idear mecanismos que protejan esta información tanto de daños físicos como de accesos inadecuados o mal intencionados. El término *seguridad* se suele referir al problema general y el término *mecanismo de protección* a los procedimientos específicos utilizados por el SO para asegurar la información del computador.

La ***seguridad*** de un sistema abarca la protección ante posibles daños físicos (incendio, terremotos, etc.) de los datos hasta el acceso indebido a los mismos (intrusos, fallos de confidencialidad, etc.). Es decir, la seguridad estudia cómo proteger la información almacenada en el sistema contra accesos indebidos o no autorizados.

Los ataques contra la confidencialidad, la integridad o la disponibilidad de recursos en un sistema deben prevenirse y solventarse mediante la política y los mecanismos de seguridad adecuados. La seguridad debe tener en cuenta eventos externos provenientes del entorno en el que opera el sistema. El SO debe ser capaz de identificar a los usuarios que acceden al sistema de manera remota o a través de un sistema distribuido.

La ***protección*** consiste en controlar y evitar o impedir que se haga un uso indebido de los recursos que están dentro del ámbito del SO. Para ello deben existir políticas y mecanismos que aseguren que los usuarios sólo acceden a sus “propios” recursos (archivos, zonas de memoria, etc.). Además, es necesario comprobar que los recursos sólo son utilizados por aquellos usuarios que tienen derechos de acceso a los mismos.

Las políticas de protección y seguridad de HW, SW y datos deben incluirse dentro del SO. En cualquier caso, el SO debe proporcionar medios para implementar la política de protección deseada por el usuario y garantizar su correcto funcionamiento.

**Objetivos o finalidades de la seguridad:**

* *Confidencialidad o Privacidad:* exige que la información sea accesible para lectura solamente por grupos autorizados.
* *Integridad:* exige que los elementos de un sistema puedan ser modificados solo por grupos autorizados.
* *Disponibilidad:* exige que los elementos estén disponibles para los grupos autorizados.

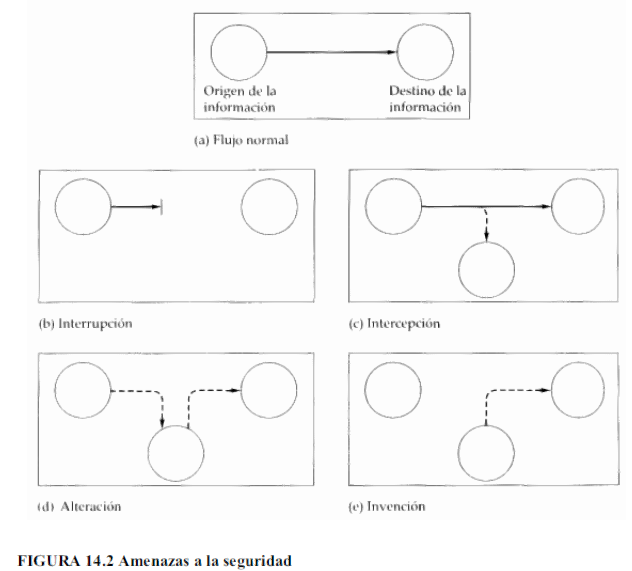
**ASPECTOS DE DISEÑO DE SO SEGUROS**

1. *Mínimo privilegio:* todos los programas y usuarios del sistema deben operar utilizando el menor conjunto de privilegios necesarios para completar la labor. Los derechos de acceso deben adquirirse solo por permiso explicito; por defecto, deberían ser "sin acceso".
2. *Ahorro de mecanismos:* los mecanismos de seguridad deben ser tan pequeños y simples como sea posible, ayudando en su verificación. Deben ser una parte integral del diseño, más que mecanismos añadidos a diseños existentes.
3. *Aceptación:* los mecanismos de seguridad no deben interferir excesivamente en el trabajo de los usuarios, mientras cumplen al mismo tiempo las necesidades de aquellos que autoricen el acceso. Deben ser fáciles de usar.
4. *Mediación total:* cada acceso debe ser comparado con la información de control de acceso incluyendo aquellos accesos que suceden fuera de la operación normal, como la recuperación y el mantenimiento.
5. *Diseño abierto:* la seguridad del sistema no debe guardar en secreto el diseño de sus mecanismos para que éstos puedan ser revisados por muchos expertos y los usuarios puedan depositar una alta confianza en ellos.

**Supervisión de Amenazas**

Los tipos de amenazas a la seguridad de sistema de computadores o una red se caracterizan mejor contemplando la función del sistema como un suministrador de información de un origen (archivo/región de memoria) a un destino (archivo/usuario). En base a esta situación, se clasifican cuatro categorías de amenazas:

1. Interrupción: se destruye un elemento del sistema o se hace inaccesible o inútil. Esta es una amenaza a la *disponibilidad*. Ejemplo: destrucción de una pieza de hardware, inutilización del sistema de gestión de archivos.
2. Intercepción: una parte no autorizada consigue acceder a un elemento. Esta es una amenaza a la *confidencialidad*. La parte no autorizada puede ser una persona, un programa o un computador. Ejemplo: interceptación de las conexiones telefónica para capturar datos de una red, copia ilícita de archivos o programas.
3. Alteración: una parte no autorizada accede y falsifica un elemento. Esta es una amenaza a la *integridad*. Ejemplo: cambio de datos en un archivo, alteración de un programa para que se comporte diferente, modificación de los mensajes transmitidos en una red.
4. Invención: una parte no autorizada inserta objetos falsos en el sistema. Esta es también una amenaza a la *integridad*. Ejemplo: inserción de mensajes falsos en una red, adición de registros en un archivo.



**PENETRACIÓN EN LOS SISTEMAS – MODALIDAD DE PENETRACIÓN**

Diferentes formas y medios:

1. **Bacteria:** programa que consume recursos del sistema reproduciéndose a sí mismo.
2. **Bomba Lógica:** lógica incrustada en un programa que, cuando se cumpla un cierto conjunto de condiciones en el sistema, ejecutará alguna función que provee a acciones no autorizadas a este.
3. **Trampilla:** punto de entrada secreto y no documentado a un programa, empleado para otorgar el acceso fuera de los métodos usuales de autentificación.
4. La utilización por parte del intruso de la cuenta de un usuario legítimo. Para conseguirlo puede usar un terminal con una sesión abierta que haya dejado el usuario legítimo u obteniendo la contraseña utilizando distintas técnicas como adivinación, ensayos de prueba y error, robo o intimidación.
5. **Troyanos:** programas que ocultan parte de su funcionalidad, frecuentemente destinada a obtener datos o derechos de acceso del usuario. Ejemplo: programa falso de "login:", idéntico en presentación al del sistema.
6. Propagación de **gusanos y virus** informáticos: El virus es parte del código de un programa y el gusano es un programa en sí mismo. El gusano carga en exceso al computador, usando para su propagación recursos desproporcionados de procesamiento y comunicación, con lo que el sistema puede denegar servicios a usuarios legítimos. El virus infectará a otros programas copiándose (cuando se arranca el programa, examina todos los programas del disco y si encuentra uno sin infectar, lo infecta añadiendo el código del virus al final del archivo del programa y sustituyendo la primera instrucción por un salto a la primera instrucción del código del virus). Por lo general también realiza actividades dañinas, como eliminar archivos o corromper los bloques de arranque del disco.
7. La inspección del sistema de archivos. Sistemas como Unix permiten, por defecto, a todos los usuarios leer los archivos. Un usuario mal intencionado podría acceder al archivo de contraseñas, copiarlo y utilizar técnicas de análisis criptográfico y métodos de prueba y error para descifrar las contraseñas. Además en estos archivos suele haber información relativa a los usuarios, como su nombre, dirección o teléfono, el ID de su sistema, programa de arranque (generalmente el intérprete de comandos que utiliza), directorios, y otras informaciones que no están codificadas.

**SEGURIDAD EN REDES**

Posibles posiciones de ataque a la seguridad

En una red, la información introducida en un terminal debe pasar por enlaces de comunicación, un controlador de grupos, una red de conmutación de paquetes (nodos y enlaces) para llegar al nodo al que está conectado el host de servicio de destino.

El ataque puede tener lugar en cualquiera de los enlaces de comunicación:

* Ataque activo: el atacante tiene que disponer de control físico sobre una parte del enlace y ser capaz de insertar y capturar las transmisiones.
* Ataque pasivo: el atacante sólo tiene que observar las transmisiones interviniendo la línea de comunicaciones.

Los procesadores de ruta también están sujetos al ataque: intentos de modificar HW o SW, obtener acceso a la memoria del procesador o supervisar ondas electromagnéticas. Son menos probables.

Ubicación de dispositivos de cifrado

El método más potente y habitual para contrarrestar las amenazas es el ***cifrado***. Hace falta decidir qué cifrar y dónde debe ubicarse el mecanismo de cifrado. Hay dos alternativas fundamentales:

* ***Cifrado de enlaces***: Cada enlace de comunicaciones se equipa en ambos extremos con dispositivos de cifrado, lo que asegura el tráfico de todos los enlaces. Esto requiere muchos dispositivos de cifrado en una red grande, pero su valor es evidente.

Desventaja: los mensajes deben ser descifrados cada vez que entran en un conmutador de paquetes para que éste pueda encaminarlo, lo que hace vulnerable al mensaje en cada conmutación.

* ***Cifrado de extremo a extremo*:** El proceso de cifrado se efectúa en los dos sistemas finales. El host de origen cifra los datos, los cuales son transmitidos sin modificación por la red hacia el host de destino, quien comparte una clave con el origen y así es capaz de descifrar los datos.

Punto débil: con este cifrado, los datos de usuario están seguros, pero la composición del tráfico no, porque las cabeceras de los paquetes se transmiten sin cifrado, para que puedan ser encaminados.

Para lograr una seguridad mayor, se necesitan ambos métodos de cifrado. Cuando se emplean ambas formas, el paquete está seguro, excepto durante el tiempo que está en la memoria de un conmutador de paquetes, en el que la cabecera del paquete está en claro.

Distribución de la clave

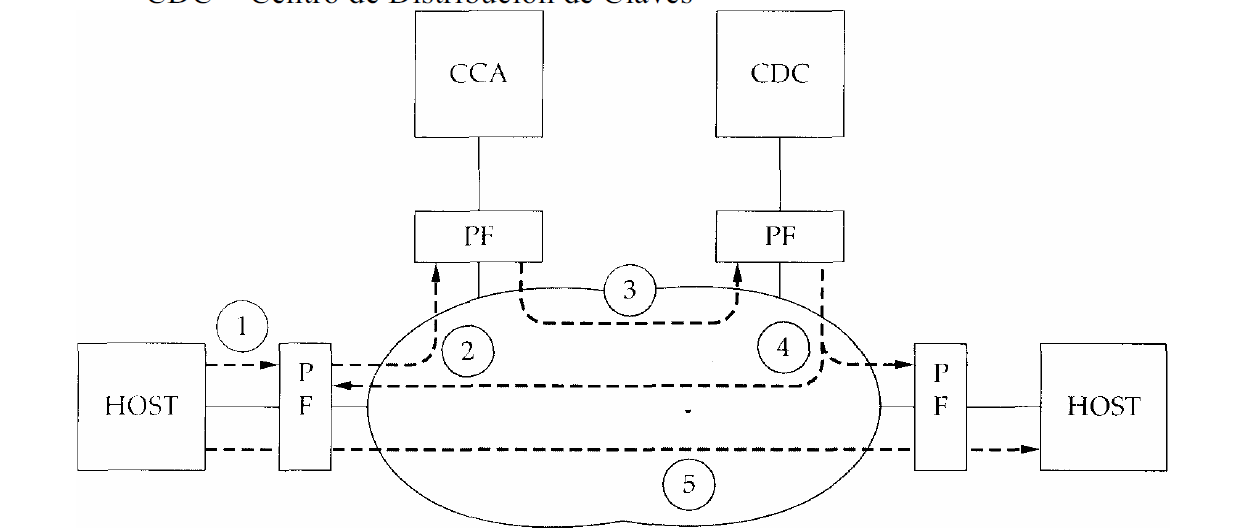
Para que el cifrado funcione, ambas partes de un intercambio deben disponer de la misma clave, que debe estar protegida del acceso de otras partes. La técnica de distribución de la clave determina la solidez de cualquier sistema de cifrado.

Existen distintas formas, para dos partes A y B:

1. A puede elegir una clave y entregarla físicamente a B.
2. Una tercera parte puede elegir una clave y entregarla físicamente a A y B.
3. Si A y B han empleado una clave recientemente, una parte puede transmitirle a la otra la nueva clave, cifrada con la antigua.
4. Si A y B tienen una conexión cifrada con una tercera parte C, C puede entregar una clave a A y B por los enlaces cifrados.

Las opciones 1 y 2 exigen la entrega manual de la clave, lo que es razonable para el cifrado de enlaces pero no para el cifrado de extremo a extremo. La opción 3 es una posibilidad para ambos métodos, pero si un atacante tiene éxito alguna vez y consigue una de las claves, todas las claves siguientes serán reveladas. Para obtener claves de cifrado de extremo a extremo, es preferible la opción 4.

Implementación que satisface la opción 4 (se ignora el cifrado de enlaces, pero podría añadirse):



Se identifican dos tipos de clave para este esquema:

* Clave de sesión: cuando los sistemas finales quieren comunicarse, establecen una conexión lógica. En ella, los datos se cifran con una clave de sesión de un solo uso, la cual se destruye al finalizar la conexión.
* Clave permanente: clave empleada por dos entidades para la distribución de las claves de sesión.

Elementos de la configuración:

* Centro de control de acceso (CCA): determina a qué sistemas se les permite comunicarse con otros.
* Centro de distribución de claves (CDC): proporciona una clave de sesión de un solo uso para una conexión permitida.
* Procesador frontal (PF): lleva a cabo el cifrado de extremo a extremo y obtiene las claves de sesión.

Pasos:

1. El host envía un paquete solicitando conexión.
2. El PF almacena el paquete y solicita al CCA permiso para la conexión y una clave de sesión.
3. El CCA aprueba la solicitud y se la ordena al CDC.
4. El CDC distribuye una clave de sesión a ambos PF.
5. Se transmiten los paquetes almacenados (cifrados con la clave de sesión).

Tanto el CCA como el CDC tienen claves permanentes únicas para su comunicación con cada PF.

Es posible variar el esquema anterior, combinando el CCA con el CDC, quitando el CCA si se quiere que los host se comuniquen a voluntad, las funciones del PF pueden incorporarse en el host, etc.

Este método de distribución automática de claves ofrece la flexibilidad y las características dinámicas necesarias para que una serie de usuarios de terminales puedan acceder a una serie de host e intercambien datos.

Otro método de distribución de claves es el *cifrado de clave pública*, el cual tiene como desventaja fundamental que sus algoritmos son mucho más complejos (frente al cifrado convencional).

Tráfico de relleno

Con el cifrado de enlaces, las cabeceras de los paquetes van cifradas, reduciendo la oportunidad de un análisis de tráfico (aunque sigue siendo posible).

Una contramedida efectiva para este ataque es el tráfico de relleno, el cual es una función que produce una salida cifrada continuamente, incluso en la ausencia de texto en claro. Se genera un flujo continuo de datos aleatorios que, en ausencia de texto en claro, se cifran y transmiten. Esto hace imposible distinguir entre flujos de datos verdaderos y ruido, lo que imposibilita deducir la cantidad de tráfico.

**PROTECCIÓN**

Los mecanismos de protección surgieron con la multiprogramación, con la intención de que los programas de cada uno de los usuarios estuvieran en la partición de memoria asignada e impedir así que los programas traspasaran sus límites a otras particiones y las dañaran. Pero la necesidad de compartir objetos tanto en MP como en la MS motivó que los mecanismos de control de acceso se hicieran más complejos.

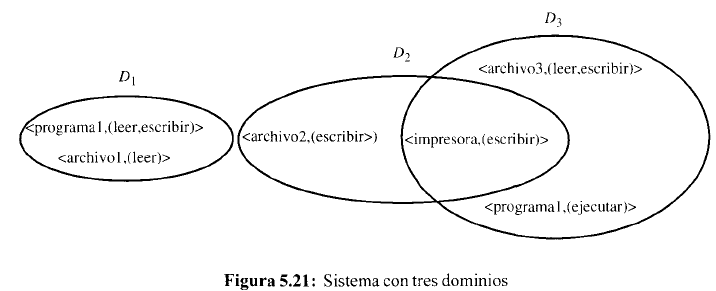
Es importante distinguir entre *política de protección* y *mecanismo de protección.* Los mecanismos dicen cómo se hará algo, mientras que la política dice qué se hará. Esta separación permite una mayor flexibilidad, ya que puede cambiar la política a lo largo del tiempo, pero manteniendo un mecanismo general. Este cambio requerirá modificar sólo algunos parámetros o tablas del sistema.

**Dominios de protección**

Un sistema de cálculo es un sistema complejo que se puede idealizar como un conjunto de procesos y objetos. Por objetos se entienden tanto las distintas unidades del computador (CPU, segmentos de memoria, discos, etc.) como las diferentes informaciones que almacenan (archivos, programas, etc.). Cada objeto es un tipo abstracto de dato con un único nombre que lo distingue de los demás objetos, y sólo es posible acceder a él por medio de operaciones útiles y bien definidas (las operaciones pueden depender del objeto en cuestión). Lógicamente, un proceso sólo debe poder acceder a aquellos recursos para los cuales está autorizado y que necesita en ese momento para completar su tarea. Este requisito se denomina *principio de la necesidad de saber,* y es útil para limitar los posibles daños que puede causar en el sistema un proceso con errores. Por ejemplo, se puede permitir que el proceso P lea el archivo A pero que no lo escriba. Para analizar los mecanismos de protección es conveniente incorporar el concepto de ***dominio de protección****.* Así cada proceso trabaja dentro de un dominio, el cual especifica los recursos a los cuales puede tener acceso. Cada dominio define un conjunto de objetos y las operaciones que se les pueden aplicar. La capacidad para ejecutar una operación sobre un objeto es un derecho de acceso. Un dominio es un conjunto de derechos de acceso, cada uno de los cuales está formado por un par de la forma:

<nombre del objeto, conjunto de sus derechos>

Ejemplo: En la imagen se muestran los derechos de acceso para cada objeto en los tres dominios (D1, *D2,* D3). Para poder leer y escribir el objeto “programa1” es necesario que un proceso se esté ejecutando en el dominio *D1,* pero para poderlo ejecutar es necesario que esté en el dominio *D3.* Por otra parte los dominios *D2* y *D3* comparten el objeto impresora con derecho de sólo escritura. En cada momento se ejecuta un proceso en algún dominio de protección.

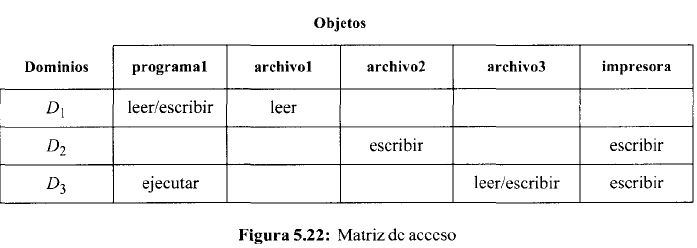


Estas ideas se pueden aclarar si se estudia el caso de Unix. En Unix el dominio de un proceso está definido por el identificador de usuario *(uid)* y del grupo *(gid).* Para un *uid* y *un gid* dados se puede elaborar una lista completa de todos los objetos a los que puede tener acceso y el tipo de acceso, es decir, una lista de archivos, dispositivos, etc., con sus correspondientes permisos de acceso.

**Matriz de Acceso o Protección**

Las relaciones entre dominios y objetos se pueden representar de forma abstracta mediante una matriz denominada **matriz** de acceso. Las filas de la matriz de acceso representan dominios y las columnas objetos.

La siguiente imagen es un ejemplo de una matriz de acceso para el mismo caso de la imagen anterior. Cada elemento de la matriz consiste en un conjunto de derechos de acceso.

Cuando se ejecuta un proceso en el dominio *D1* puede leer y escribir programa1 y sólo leer archivo1. En el dominio *D3* se puede ejecutar programa1 pero no se puede leer ni escribir en él, sin embargo se puede leer y escribir en archivo3 y escribir en impresora. En el dominio *D2* solo se puede escribir en el archivo2 y en la impresora.

Con esta matriz y el número del dominio, el sistema puede determinar qué tipo de acceso se permite a un objeto específico. El problema es cómo realizar de una forma eficaz esta matriz.

Generalmente, la matriz de acceso es una matriz dispersa, con muchos elementos vacíos. Por ello una forma sencilla de realizarla es mediante una *tabla global*, consistente en tripletas ordenadas (dominio, objeto, derechos). Cuando un proceso ejecuta una operación *P* para un objeto *O* dentro de un dominio *D,* se busca en la tabla global la tripleta (D, O, derechos\_acceso), donde PE derechos\_acceso. Si se encuentra esta tripleta se permite la operación y el proceso puede continuar, sino se produce una condición de error.

Inconvenientes: la tabla suele ser grande y no se puede conservar en MP, por lo que requiere operaciones adicionales de E/S. Además, no se pueden aprovechar las agrupaciones de objetos o dominio: por ejemplo, si todos pueden leer un objeto éste deberá tener una entrada en cada dominio.

Dos métodos prácticos que se suelen utilizar están basados en almacenar la matriz por columnas *(lista de accesos)* o por filas *(lista de capacidades).*

**Lista de accesos**

A cada objeto se le asocia una lista ordenada con todos los dominios que pueden tener acceso al objeto y la forma de dicho acceso.

Ejemplo para los objetos del ejemplo anterior:

programa 1: (D1, leer-escribir), (D3, ejecutar)

archivo 1: (D¡, leer)

archivo2: (D2, escribir)

archivo3: *(D3,* leer-escribir)

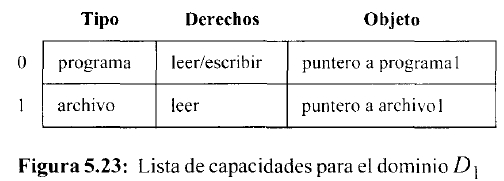
impresora: (D2, escribir), *(D3,* escribir)

El principal inconveniente de las listas de accesos es el retardo que se provoca con la búsqueda para verificar la autoridad de un sujeto para acceder al objeto solicitado.

Para evitar búsquedas de listas potencialmente muy largas de usuarios autorizados y ahorrar espacio de almacenamiento, algunos sistemas dividen a los usuarios en grupos y sólo almacenan los derechos de acceso de los grupos. Este esquema ahorra almacenamiento y agiliza el procesamiento reduciendo la flexibilidad y limitando el número de dominios.

**Lista de capacidades**

A cada dominio (o sujeto) se le asocia una lista de objetos a los cuales puede tener acceso, junto con una indicación de las operaciones permitidas. Ejemplo para el dominio *D1* del ejemplo anterior:



La lista de capacidades está asociada a un dominio, pero un proceso que se ejecuta en ese dominio no puede acceder a ella directamente. En cambio, la lista de capacidades es un objeto protegido, mantenido por el SO y al que el usuario sólo puede tener acceso de forma indirecta.

La protección mediante capacidades se basa en que nunca se permite que una capacidad se mueva al espacio de direcciones accesibles por un proceso de un usuario. Manteniendo las capacidades seguras, los objetos a los que protegen también están seguros frente a un acceso no autorizado. Esta idea de punteros inherentemente protegidos, desde el punto de vista del usuario, proporciona una base de protección ampliable también al nivel de aplicaciones.

Uno de los inconvenientes de las listas de capacidades es la revocación de acceso a un objeto, ya que el sistema tiene que determinar todas las capacidades existentes para cierto objeto y eliminarlas, puesto que pueden estar almacenadas en las listas de capacidades a lo largo de todo el disco. Una forma de eliminar este problema es hacer que las capacidades apunten a un objeto indirecto en vez de al objeto real. La forma de invalidar la capacidad es romper el enlace entre el objeto indirecto y el real.

**Estructuras de Protección Dinámicas**

Las matrices de acceso vistas hasta el momento, en las que no cambian los derechos en cada dominio durante su ejecución, son un ejemplo de *Estructuras de Protección Estáticas*. Con el fin de ofrecer flexibilidad y de implementar eficientemente la protección, un SO debe soportar cambios en los derechos de acceso. Para esto se requiere implementar alguna estructura de protección dinámica.

Básicamente se requieren cuatro nuevos derechos de acceso: ***Copia, Cambio, Propietario y Control.***

**Derecho de acceso copia**

Da la facultad a un proceso de copiar derechos existentes en un dominio hacia otro dominio para el objeto en cuestión. O sea, este derecho genera copias en columnas.

En general se podría hablar de 3 variantes del derecho "copia":

Copia limitada: La copia no incluye el derecho "Copia", con lo que no se podrán hacer copias sucesivas del derecho.

Copia completa: La copia incluye el derecho "Copia".

Translación: El derecho en cuestión junto con el derecho copia se eliminan del dominio original y se coloca en el nuevo dominio. Esta es una forma básica de retirar derechos de un dominio para asignarlo a otro.

**Derecho de acceso cambio**

Indica la posibilidad de un proceso para cambiarse de un dominio a otro. La operación cambio actúa sobre dominios, o sea, en este caso los dominios son los objetos. Entonces, para considerar este derecho de acceso se deberán incluir los dominios como objetos adicionales en la matriz de acceso.

**SISTEMAS DISTRIBUIDOS**

Aparecen los *sistemas distribuidos*, en contraste con los sistemas centralizados. Los sistemas distribuidos necesitan un software distinto al de los sistemas centralizados.

Los SO para sistemas distribuidos se siguen desarrollando.

En general se consideran sistemas distribuidos, en sentido amplio, a los sistemas en que:

* Existen varias cpu conectadas entre sí.
* Las distintas cpu trabajan de manera conjunta.

Distinguir entre:

* *Sistemas distribuidos***:** están diseñados para que muchos usuarios trabajen en forma conjunta.
* *Sistemas paralelos***:** están diseñados para lograr la máxima rapidez en un único problema.

**Ventajas de los Sistemas Distribuidos**

Con respecto a los Centralizados:

* Más económico. Generalmente tienen en potencia una proporción precio/desempeño mucho mejor que la de un único sistema centralizado.
* Mayor confiabilidad. Al distribuir la carga de trabajo en muchas máquinas, la falla de una de ellas no afectará a las demás.
* Crecimiento incremental. Podrían añadirse procesadores al sistema, permitiendo un desarrollo gradual según las necesidades.

Con Respecto a las PC Independientes:

* Satisfacen la necesidad de muchos usuarios de compartir ciertos datos, programas y periféricos costosos.
* Mejor comunicación entre las personas.
* Mayor flexibilidad. La carga de trabajo se puede difundir entre las máquinas disponibles en la forma más eficaz según el criterio adoptado.

**Desventajas de los Sistemas Distribuidos**

* El diseño, implantación y uso del SW distribuido presenta numerosos inconvenientes.
* Se deben considerar problemas de redes de comunicaciones debidos a pérdidas de mensajes, saturación en el tráfico, expansión, etc.

Se debe garantizar la seguridad de los datos ya que es muy fácil de compartir.

Las ventajas superan a las desventajas, si estas últimas se administran seriamente.

**CONCEPTOS DE HARDWARE**

Todos los sistemas distribuidos constan de varias cpu, organizadas de diversas formas, especialmente respecto de:

* La forma de interconectarlas entre sí.
* Los esquemas de comunicación utilizados.

Existen diversos esquemas de clasificación para los sistemas de cómputos con varias cpu. Uno de los más conocidos es la “Taxonomía de Flynn”:

* Considera como características esenciales el número de flujo de instrucciones y el número de flujos de datos.
* La clasificación incluye equipos **SISD, SIMD, MISD** y **MIMD.**

SISD: Un flujo de instrucciones y un flujo de datos. Poseen un único procesador.

SIMD: Un flujo de instrucciones y varios flujos de datos. Ordenar varios procesadores con una unidad de instrucción.

MISD: Un flujo de varias instrucciones y un solo flujo de datos. No se presenta en la práctica.

MIMD: Un grupo de computadoras independientes, cada una con su propio contador del programa, programa y datos. *Todos los sistemas distribuidos son de este tipo.*

Un avance sobre la clasificación de Flynn incluye la división de las computadoras MIMD en dos grupos:

* Multiprocesadores: poseen memoria compartida. Los distintos procesadores comparten el mismo espacio de direcciones virtuales. Fuertemente acoplados.
* Multicomputadoras: no poseen memoria compartida. PC independientes.

Cada una de las categorías indicadas se puede clasificar según la arquitectura de la red de interconexión en:

* Esquema de bus: Existe una sola red, bus, cable u otro medio que conecta todas las máquinas.
* Esquema con conmutador: No existe una sola columna vertebral de conexión. Hay múltiples conexiones y varios patrones de conexionado.

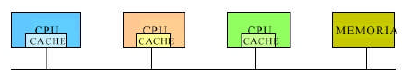
Otro aspecto de la clasificación considera el acoplamiento entre los equipos:

* Sistemas fuertemente acoplados: Poco retraso al enviar un mensaje y alta velocidad. Generalmente se los utiliza como sistemas paralelos.
* Sistemas débilmente acoplados: El retraso de los mensajes es grande y la velocidad es baja. Generalmente se los utiliza como sistemas distribuidos.

Generalmente los multiprocesadores están más fuertemente acoplados que las multicomputadoras.

* **Multiprocesadores con Base en Buses**

Constan de cierto número de cpu conectadas a un bus común, junto con un módulo de memoria.



La memoria presenta la propiedad de coherencia**:** Las modificaciones hechas por una cpu se reflejan de inmediato en las subsiguientes lecturas de la misma o de otra cpu.

Desventaja: El bus tiende a sobrecargarse y el rendimiento a disminuir drásticamente. La solución es añadir una memoria caché de alta velocidad entre cada cpu y el bus:

* El caché guarda las palabras de acceso reciente.
* Todas las solicitudes de la memoria pasan a través del caché. Si la palabra solicitada se encuentra en el caché, no se hace solicitud alguna al bus.
* Si el caché es lo bastante grande, la cantidad de tráfico en el bus por cada cpu disminuirá drásticamente y permite incrementar el número de cpu.

Problema de cachés: Incoherencia de la memoria, ya que los cambios realizados sobre un dato en caché no se guardan en la memoria. La solución sería diseñar las caché de tal forma que cuando una palabra sea escrita al caché, también sea escrita a la memoria (caché de escritura) y, además, monitorear el bus para actualizar los datos en las cachés (cachés monitores).

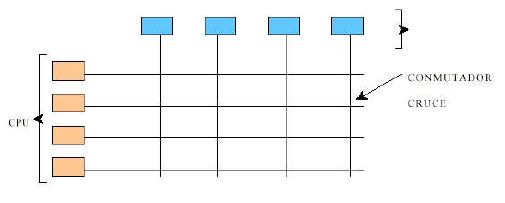
Un diseño con cachés monitores y de escritura es coherente e invisible para el programador, por lo que es muy utilizado en multiprocesadores basados en buses.

* **Multiprocesadores con Conmutador**

Esquema de conmutador cruceta: se divide la memoria en módulos y se conecta cada cpu y cada módulo con un conmutador de cruce

*Ventaja*: todas las cpu conectadas pueden tener acceso a la memoria al mismo tiempo, aunque no a la misma memoria simultáneamente.

*Desventaja*: alto número de conmutadores.



MEMORIA

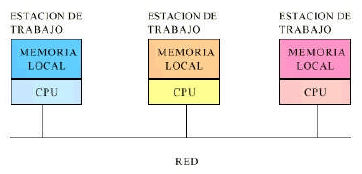
Esquemas según sistemas jerárquicos: cada cpu tiene asociada cierta memoria local. El acceso será muy rápido a la propia memoria local y más lento a la memoria de las demás cpu.

* **Multicomputadoras con Base en Buses**

Cada cpu tiene una conexión directa con su propia memoria local, es decir, no se comparte la memoria.

Un problema importante es la forma en que las cpu se comuniquen entre sí. El tráfico es solo entre una cpu y otra; el volumen de tráfico será menor que si se utilizara multiprocesador con base en buses.

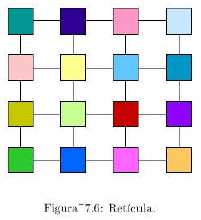
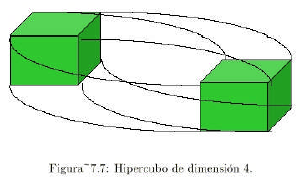
Topológicamente es un esquema similar al del multiprocesador basado en un bus. Consiste generalmente en una colección de estaciones de trabajo en una LAN.



* **Multicomputadoras con Conmutador**

Cada cpu tiene acceso directo y exclusivo a su propia memoria particular.

Existen diversas topologías, las más comunes son la **retícula** y el **hipercubo**.

*Los multiprocesadores basados en bus, incluso con cachés, están limitados por la capacidad del bus (hasta 64 procesadores). Para pasar este límite, la solución ideal para sistemas grandes son las conexiones hipercubo.*

**CONCEPTOS DE SW**

La importancia del SW supera frecuentemente a la del HW. La imagen que un sistema presenta queda determinada en gran medida por el SW del SO.

Se puede clasificar a los SO en dos tipos:

* Débilmente acoplados.
* Fuertemente acoplados.

El software débilmente acoplado de un sistema distribuido permite que las máquinas y usuarios sean independientes entre sí en lo fundamental. Facilita que interactúen en cierto grado cuando sea necesario. Los equipos individuales se distinguen fácilmente.

Combinando los distintos tipos de HW distribuido con SW distribuido se logran distintas soluciones:

* **Sistemas Operativos de Redes**

SW débilmente acoplado en HW débilmente acoplado. Cada usuario tiene una estación de trabajo para su uso exclusivo: tienen su propio SO por lo que la mayoría de los requerimientos se resuelven localmente.

Es muy utilizado en una red de estaciones de trabajo conectadas mediante una LAN.

Es posible que un usuario se conecte de manera remota con otra estación de trabajo.

Las redes también disponen de un comando de copiado remoto de archivos de una máquina a otra. Una mejora de esto consiste en un *sistema de archivos global compartido*, accesible desde todas las estaciones de trabajo. Este **sistema de archivos** es soportado por una o varias máquinas, llamadas **servidores**. Estos aceptan solicitudes de los programas de usuarios, las examinan, ejecutan y envían la respuesta deregreso. Los programas se ejecutan en las máquinas no servidoras, llamadas **clientes**.

El SO debe controlar las estaciones de trabajo individuales, a los servidores de archivo y encargarse de la comunicación entre los servidores.

**NFS: Network File System**

El SO de red más conocido y aceptado es el NFS. Este solo trata el sistema de archivos y no hace referencia a otros aspectos, como la ejecución de un proceso.

Cada computadora puede ejecutar su propio SO, solo se dispone de un sistema compartido de archivos y el tráfico cliente - servidor debe obedecer los protocolos NFS.

La Arquitectura de NFS

La idea fundamental es permitir que una colección arbitraria de clientes y servidores compartan un sistema de archivos común. Generalmente están en la misma LAN, aunque se puede ejecutar NFS en una WAN.

NFS permite que cada máquina sea un cliente y un servidor al mismo tiempo. Los servidores soportan físicamente el sistema de archivos compartido, mientras que los clientes sólo lo utilizan. Los servidores exportan uno o varios de sus directorios para el acceso por parte de clientes remotos. Los clientes pueden montar los directorios exportados como un directorio más en su jerarquía de directorios.

Por lo tanto, los directorios de distintas máquinas se pueden leer y escribir como si fueran locales y forman parte de la jerarquía de directorios.

Protocolos de NFS

NFS define dos **protocolos cliente – servidor** para soportar un sistema heterogéneo en donde los clientes y servidores podrían ejecutar distintos SO en HW diverso. (Protocolo: conjunto de solicitudes y respuestas entre los clientes y los servidores)

Un protocolo de NFSmaneja el montaje:

* Un cliente puede enviar el nombre de una ruta de acceso a un servidor y solicitar el permiso para montar ese directorio en alguna parte de su jerarquía de directorios.
* Algunos SO soportan la alternativa del automontaje: permite que un conjunto de directorios remotos quede asociado con un directorio local.
* NFS no da soporte a la duplicación de archivos o directorios.

Otro protocolo de NFSes para el acceso a los directorios y archivos:

* Para leer, escribir y tener acceso a los atributos de los directorios y archivos, los clientes pueden enviar mensajes a los servidores.
* No es necesario abrir ni cerrar los archivos para utilizarlos. La ventaja de este esquema es que el servidor no tiene que recordar las conexiones abiertas, por lo que, si falla, al volver a arrancar no pierde información acerca de los archivos abiertos, porque no existen.

Implantación de NFS

La implantación del código del cliente y el servidor es independiente de los protocolos NFS.

Consiste en tres capas:

* Llamadas al sistema: open, read, write, close.
* Sistema Virtual de Archivos (VSF): mantiene una tabla con una entrada por cada archivo abierto.
* Capa de servicios NFS: implementa el protocolo NFS.
* **Sistemas Realmente Distribuidos**

SW fuertemente acoplado en HW débilmente acoplado. Multicomputadoras.

Un sistema distribuido es aquel que se ejecuta en una colección de máquinas sin memoria compartida, pero que aparece ante sus usuarios como una sola computadora. Crean la ilusión de que toda la red de computadoras es un solo sistema de tiempo compartido, en vez de una colección de máquinas diversas (imagen de un único sistema).

Los usuarios pueden acceder a los recursos remotos de igual manera que a los locales. El SO distribuido se encargará de la migración de datos y procesos de una máquina a otra.

Algunas de las características de los sistemas distribuidos son las siguientes:

* Debe existir un mecanismo de comunicación global entre los procesos: cualquier proceso debe poder comunicarse con cualquier otro.
* Debe existir un esquema global de protección.
* La administración de procesos debe ser la misma en todas partes.
* Se debe tener una misma interfaz de llamadas al sistema en todas partes: núcleos idénticos en todas las cpu.
* Es necesario un sistema global de archivos.

En un sistema distribuido no hay memoria ni relojes comunes, por lo que es imposible determinar qué suceso ocurrió primero. Para ello se utiliza la técnica de marcación de tiempo que consiste en ordenar los sucesos sin usar relojes físicos (sucesos: acciones - que un proceso entre o salga de su región crítica), sino según la transmisión de mensajes y un contador que funciona como reloj.

* **Sistemas de Multiprocesador con Tiempo Compartido**

SW fuertemente acoplado en HW fuertemente acoplado. Multiprocesadores.

Operan como un sistema de tiempo compartido, pero con varias cpu en vez de una sola. Se corresponde con la imagen de un único sistema. Ninguna cpu tiene memoria local, hay una memoria global compartida con el SO y los programas.

Existe de una sola cola para ejecución, con todos los procesos listos, almacenada en la memoria compartida.

El planificador (de procesos) del SO se ejecuta como una “región crítica”, con ello se evita que dos cpu elijan el mismo proceso para su ejecución inmediata.

Si todas las cpu están inactivas en espera de E/S y un proceso está listo para su ejecución es conveniente asignarlo a la cpu que se utilizó por última vez (para ese proceso). Si un proceso se bloquea en espera de E/S, el SO puede suspenderlo o dejarlo en “espera ocupada”

Generalmente se dispondrá de un sistema de archivos tradicional, con un único caché: Globalmente considerado es similar al sistema de archivos de un único procesador.

**ASPECTOS DEL DISEÑO DE SO DISTRIBUIDOS**

* **Transparencia**

Lograr la imagen de un único sistema. Los usuarios deben percibir la colección de máquinas como un sistema único, sin distinción entre recursos locales y remotos.

Existen distintos tipos de transparencia en un sistema distribuido: de la localización, de la migración, de copia, con respecto a la concurrencia y con respecto al paralelismo.

* **Flexibilidad**

Dos concepciones:

* *Sistemas con núcleo monolítico*: Cada máquina debe ejecutar un núcleo tradicional que proporcione la mayoría de los servicios, al que se le agregan capacidades de red e integración de servicios remotos. Ventaja: mayor rendimiento.
  + *Sistemas basados en un micronúcleo*: cuentan con los servicios mínimos, consistentes en un mecanismo de comunicación entre procesos, administración de la memoria, planificación y gestión de los procesos de bajo nivel y gestión de E/S de bajo nivel. El resto de los servicios se implementan como servidores a nivel de usuario.

Ventajas: más flexibles, muy modulares, permiten instalar y depurar nuevos servicios fácilmente.

* **Confiabilidad**

El sistema debe ser tolerante a fallos. Si una máquina falla, otra debe encargarse del trabajo. Algunos aspectos relacionados son la disponibilidad y seguridad del sistema.

* **Desempeño**

La ejecución de una aplicación en un sistema distribuido debe ser igual a su ejecución en un único procesador, pero esto es difícil de lograr.

* **Escalabilidad**

El sistema debe tener capacidad de crecer sin problemas y permitir una elevada carga de servicio, adaptándose al crecimiento del número de usuarios y admitiendo la integración de nuevos recursos adicionales de una forma sencilla.