FUNDAMENTOS DE LOS SISTEMAS OPERATIVOS

1. EL SISTEMA OPERATIVO 1
1.1. FUNCIONES DEL SISTEMA OPERATIVO 1
1.2. ESTRUCTURA DE UN SISTEMA OPERATIVO 3
1.3. UTILIZACIÓN DEL SISTEMA OPERATIVO 4
1.4. CLASIFICACION DE LOS SISTEMAS OPERATIVOS 5
Por los servicios ofrecidos: 5
Por la forma de ofrecer los servicios: 5
Por su disponibilidad: 6
2. TIPOS DE APLICACIONES Y TIPOS DE LICENCIA 7
3. LOS GESTORES DE ARRANQUE 7
4. GESTIÓN DEL PROCESADOR 8
4.1. LOS PROCESOS Y LA PLANIFICACIÓN 8
4.2. GESTIÓN DE PROCESOS 8
4.3. ESTADOS DE UN PROCESO 9
4.4. CAMBIO DE PROCESO 10
4.5. PLANIFICACIÓN DEL PROCESADOR 11
Tiempo Compartido 11
4.6. ALGORITMOS DE PLANIFICACIÓN 11
Multiprogramación clásica 11
Planificación por torneo, tiempo paralelo o turno rotatorio 11
Planificación por prioridad. 12
Lista de espera con intervalos múltiples. 12
Prioridad al más corto. 12
5. GESTIÓN DE LA MEMORIA 13
Reubicación. 13
Protección. 14
Compartición. 14
5.1. TÉCNICAS DE GESTIÓN DE MEMORIA 15
5.1.1 Particiones Estáticas 15
5.1.2 Particiones Dinámicas 15
5.1.3 Paginación 16
5.1.4 Segmentación 16
5.2. MEMORIA VIRTUAL 17
6. GESTIÓN DE LAS INTERRUPCIONES. 18
7. GESTIÓN DE LA E/S 19
7.1. TÉCNICAS DE E/S 19
E/S Programada 19
E/S Dirigida por interrupciones 19
DMA 20
7.2. ALMACENAMIENTO INTERMEDIO DE LA E/S: *BUFFERING* 20

8. LENGUAJES DE PROGRAMACIÓN 21
8.1. LENGUAJES DE BAJO NIVEL. 21
8.1.1 Código máquina. 21
8.1.2 Lenguaje ensamblador. 21
8.2. LENGUAJES DE ALTO NIVEL. 22
8.3. LA TRADUCCIÓN. 23
8.3.1 Interpretación. 23
8.3.2 Compilación. 24
Compilación del fuente. 24
Encuadernación o linkaje. 24
El debugger 24
8.3.3 Traducción mixta. 24
9. CARGA Y MONTAJE. 25
9.1. MONTAJE 25
9.1.1 Editor de montaje. 25
9.1.2 Montador dinámico. 26
9.2. CARGA 27
9.2.1 Carga absoluta. 27
9.2.2 Carga reubicable. 27
9.2.3 Carga dinámica en tiempo de ejecución. 28
10. SEGURIDAD. 29
10.1. AMENAZAS A LA SEGURIDAD. 29
10.1.1 Tipos de amenazas. 29
10.1.2 Amenazas a cada uno de los elementos del sistema. 29
Hardware. 29
Software. 29
Datos. 30
Redes y líneas de comunicaciones. 30
10.2. PRINCIPIOS DE DISEÑO DE LAS MEDIDAS DE SEGURIDAD. 31
10.2.1 Protección. 31
Protección de memoria. 31
Control de acceso orientado al usuario. 31
Control de acceso orientado a los datos. 32
10.2.2 Intrusos. 32
10.2.3 Virus. 32
Ciclo de los virus. 33
Tipos de virus. 33
Camuflaje de los virus. 34
10.2.4 Los antivirus. 34
Las vacunas. 34
Antivirus residentes o centinelas. 35
Los virus e Internet. 35
Direcciones útiles. 36

1. EL SISTEMA OPERATIVO

El sistema operativo (SO) es un conjunto de programas que gestiona los recursos del sistema y optimiza su uso, actuando como intermediario entre el usuario y el hardware. Puede considerarse el sistema operativo desde dos puntos de vista:

- **SO como interfaz usuario-máquina**: Se encarga de que el usuario pueda abstraerse de las complejidades de funcionamiento del sistema, es una capa compleja entre el hardware y el usuario, que facilita herramientas adecuadas para realizar tareas informáticas, abstrayéndole de los complicados procesos necesarios para llevarlas a cabo.
- **SO como administrador de recursos**: debe repartir los recursos entre los distintos programas y y entre los usuarios de forma óptima, evitando tiempos muertos, controlando memoria, etc., pero como programa, el SO deberá cumplir los requisitos que el resto de programas que el usuario decida ejecutar, con la excepción de que tendrá más privilegios siendo el único con acceso a ciertos recursos del sistema

Los **objetivos** que debe cumplir el sistema operativo son los tres siguientes:

- **Eficiencia**: Permite que los recursos de un sistema informático se aprovechen al máximo, aumentando el rendimiento global del ordenador, por ejemplo pasando a otro programa cuando el que se está ejecutando se bloquea.
- **Comodidad**: Un sistema operativo facilita el uso del ordenador y debe hacerlo de la manera más cómoda para el usuario.
- **Capacidad de evolución**: Un sistema operativo debe construirse de modo que permita el desarrollo efectivo, la verificación y la introducción de nuevas funciones en el sistema sin interferir en los servicios que brinda, aprovechando las actualizaciones de nuevos tipos de hardware y ofreciendo nuevos servicios al usuario y la corrección de posibles fallos.

1.1. FUNCIONES DEL SISTEMA OPERATIVO

El SO es un programa más cuyas instrucciones ha de ejecutar el procesador. El SO dirige al sistema en el uso de los recursos y en el control de tiempo de ejecución asignado a cada programa.

Una parte del SO siempre está en memoria interna (esta parte se llama núcleo o kernel) e incluye las funciones utilizadas con mayor frecuencia. En un momento dado en la memoria puede haber otras partes del SO, además de programas y datos del usuario. La asignación de memoria también es controlada por el SO y el hardware de control de memoria del procesador.

El SO también decide cuándo puede utilizarse un dispositivo de E/S, y controla el acceso y utilización de archivos.

1. **Control de la ejecución de los programas (**gestión de procesos y gestión de interrupciones**).**

   Un proceso es una instancia de un programa que se está ejecutando, por lo que un mismo programa puede generar varios procesos. Estos procesos necesitan ciertos recursos, incluyendo tiempo de CPU, memoria, archivos y dispositivos de E/S.

   El SO debe crear y terminar los procesos, así como suspenderlos y reanudarlos cuando sea necesario, asignándoles tiempo de procesador. Además de esto debe realizar el cambio de proceso de forma adecuada guardando toda la información del proceso que se suspende para poder seguir desde el mismo punto cuando se reanude.

   Además, el SO debe proporcionar los mecanismos necesarios para la sincronización y comunicación entre procesos estableciendo prioridades cuando diferentes procesos soliciten el mismo recurso.

2. **Administración de periféricos** (gestión de la entrada-salida)

   El SO coordina y manipula los dispositivos conectados al ordenador. Cada dispositivo de E/S requiere un conjunto propio y peculiar de instrucciones y señales de control para su funcionamiento. El S.O. tiene en cuenta estos detalles de modo que el programador y el usuario puedan manejar el dispositivo de forma simple.

   - Gestión de Almacenamiento Secundario
     - Un **archivo** es una colección de información con nombre. Es la entidad básica de almacenamiento persistente.
     - El sistema de archivos suministra primitivas para manipular archivos y directorios: crear, borrar, leer, etc. También realiza la correspondencia entre archivos y su almacenamiento secundario abstrayendo al programador y al usuario de la naturaleza del dispositivo y del medio de almacenamiento y codificación de los datos.

3. **Gestión de permisos y de usuarios**

   Asigna los permisos de acceso a los distintos recursos (impresoras, carpetas, etc.).

4. **Administración de memoria**

   El S.O. asigna el espacio de direcciones a los procesos. Para ello debe gestionar la memoria de forma que divida toda la disponible entre el propio S.O. (recordemos que es un programa que se está ejecutando) y el resto de procesos de usuario. El S.O. debe mantener la pista de la de memoria utilizada actualmente y quién la usa. Debe además decidir cuanta memoria asignar a cada proceso, y cuando debe ser retirado de memoria un proceso.

5. **Funciones de soporte a las anteriores**

   - Intérprete de órdenes: Programa o proceso que maneja la interpretación de órdenes del usuario desde un terminal, o desde un archivo de órdenes, para acceder a los servicios del SO. Puede ser una parte estándar del SO (p. ej. el command.com de MS-DOS) o bien, un proceso no privilegiado que hace de interfaz con el usuario. Esto permite la sustitución de un intérprete por otro (p. ej. en Unix tenemos csh, bash, ksh, etc.).
   - Tratamiento de las interrupciones.
   - Detección y respuesta a errores: Cuando está funcionando un sistema pueden producirse errores internos y externos del hardware (errores de memoria, fallos en dispositivos, etc.) y/o errores de software (desbordamiento en una operación, acceso a zonas de memoria prohibidas o incapacidad del S.O. para satisfacer la solicitud de una aplicación). En cada caso el S.O. debe dar una respuesta que elimine la condición de error con el menor impacto posible. En las aplicaciones en ejecución, la respuesta puede ir desde terminar el programa que produjo el error hasta reintentar la operación o simplemente informar de tal error a la aplicación.
   - Estadísticas: Un buen S.O. debe recoger estadísticas de utilización de los diversos recursos y supervisar los parámetros de rendimiento totales como puede ser el tiempo de respuesta ante la solicitud de una aplicación. Esta información puede resultar muy útil para mejoras futuras encaminadas a aumentar el rendimiento del sistema.
   - Seguridad y protección del acceso al sistema: El S.O. controla el acceso al sistema informático como un todo y a sus recursos específicos. Puede brindar protección tanto a los recursos como a los datos ante usuarios no autorizados, y puede resolver conflictos en la propiedad de recursos.
   - Creación de programas: Editores, depuradores (debugger), compiladores y linkadores ayudan en su tarea al programador. Normalmente estos servicios se hallan en programas de utilidad que no forman realmente parte del S.O. pero que son accesibles a través de él.

1.2. ESTRUCTURA DE UN SISTEMA OPERATIVO

Si estudiamos los sistemas operativos atendiendo a su estructura interna, veremos que existen dos tipos fundamentales: los sistemas de estructura monolítica y de estructura jerárquica.

Estructura Monolítica

En los sistemas operativos de estructura monolítica el sistema operativo está formado por un único programa dividido en rutinas, en donde cualquier parte del sistema operativo tiene los mismos privilegios que cualquier otra. Cada una de las rutinas que forman el sistema operativo puede llamar a las demás cada vez que así lo requiera ya que todas son visibles entre ellas. El programa puede tener un tamaño considerable, y deberá ser recompilado por completo al añadir una nueva funcionalidad. Un error en una rutina puede propagarse a todo el núcleo. Todos sus componentes se encuentran integrados en un único programa que se ejecuta en un único espacio de direcciones. En este tipo de sistemas, todas las funciones que ofrece el sistema operativo se ejecutan en modo supervisor.

Estos sistemas operativos han surgido, normalmente, de sistemas operativos sencillos y pequeños a los que se les ha ido añadiendo un número mayor de funcionalidades. Esto les ha hecho evolucionar y crecer hasta convertirlos en programas grandes y complejos formados por muchas funciones situadas todas ellas en un mismo nivel.

El problema que plantean este tipo de sistemas radica en lo complicado que es modificar el sistema operativo para añadir nuevas funcionalidades y servicios. En efecto, añadir una nueva característica implica la modificación de un gran programa, compuesto por miles de líneas de código fuente y funciones.

Estructura Jerárquica

A medida que fueron creciendo las necesidades de los usuarios y se perfeccionaron los sistemas, se hizo necesaria una mayor organización del software del sistema operativo, donde una parte del sistema contenía subpartes y esto organizado en forma de niveles. Se dividió el sistema operativo en pequeñas partes, de tal forma que cada una de ellas estuviera perfectamente definida y con un claro interfaz con el resto de elementos. Se constituyó una **estructura jerárquica o de niveles o en capas** en la que se basan prácticamente la mayoría de los sistemas operativos actuales. Otra forma de ver este tipo de sistema es la denominada de anillos concéntricos o "*rings*"

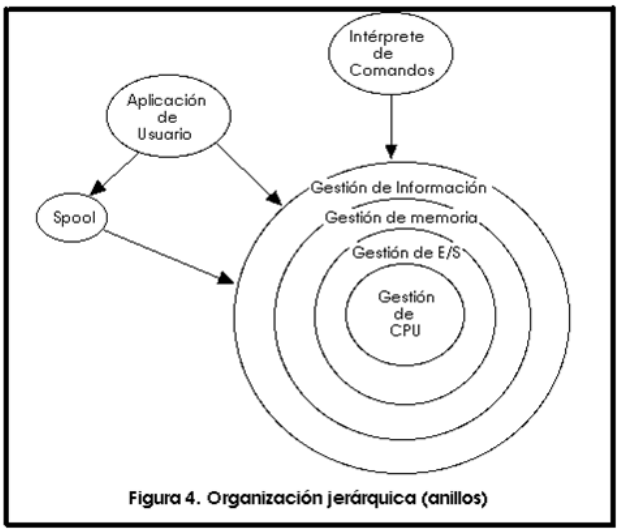


Figura 4. Organización jerárquica (anillos)

En el sistema de capas, cada capa es una máquina más abstracta para la capa superior. Cada una tiene una apertura, conocida como puerta o trampa (*trap*), por donde pueden entrar las llamadas de las capas inferiores. De esta forma, las zonas más internas del sistema operativo o núcleo del sistema estarán más protegidas de accesos indeseados desde las capas más externas. Las

capas más internas serán, por tanto, más privilegiadas que las externas. Estos niveles son totalmente invisibles para el usuario y se puede decir que son los componentes internos del sistema operativo.

La división en capas no es la misma en todos los sistemas, así por ejemplo en **Windows NT** (2000, XP y posteriores**),** los niveles en que se divide el sistema operativo son los cuatro siguientes:

- **Capa de abstracción de hardware** (HAL, *Hardware Abstraction Layer*): Establece una correspondencia entre las órdenes y respuestas genéricas del hardware y aquellas que son propias de una plataforma específica, como un Intel 486 o un Pentium, un Power PC de *Motorola* o un procesador Alpha de *Digital Equipment Corporation*. La HAL hace que el bus del sistema de cada máquina, el controlador de DMA, el controlador de interrupciones, los relojes de sistema y el módulo de memoria parezcan los mismos para el núcleo.
- **Núcleo**: Consta de las componentes más usadas y fundamentales del sistema operativo. El núcleo administra la planificación de procesos, la gestión de excepciones (cepos) e interrupciones, la sincronización de multiprocesadores y la gestión de memoria virtual. En realidad se trataría de un microkernel (micronúcleo[1]).
- **Subsistemas**: Incluyen varios módulos con funciones específicas que hacen uso de los servicios básicos proporcionados por el núcleo y se ejecutan como procesos de usuario. Por ejemplo: proceso *log-on* y subsistema de seguridad.
- **Servicios del sistema**: Ofrece una interfaz al software en modo usuario.

En **UNIX**, el hardware básico está rodeado por el software del S.O. que se llama a menudo núcleo del sistema o, simplemente núcleo (*kernel*), para realzar su aislamiento de las aplicaciones y de los usuarios. Sin embargo, UNIX viene equipado con una serie de servicios de usuario e interfaces que se consideran parte del sistema. Estos pueden agruparse en un *shell*[2], algún otro software de interfaz y los componentes del compilador de C. La capa exterior está formada por las aplicaciones de los usuarios y una interfaz de usuario con el compilador C.

1.3. UTILIZACIÓN DEL SISTEMA OPERATIVO

Debemos tener en cuenta también que externamente un S.O. se manifiesta a través de la interfaz de usuario, distinguiéndose a este respecto dos tipos:

- **Modo gráfico**: La interfaz está compuesta por un sistema de ventanas y menús desplegables que incluyen pequeños dibujos aclaratorios o iconos. Estos iconos pueden proporcionar un acceso directo a mandatos o aplicaciones. Esta interfaz es más compleja en su diseño pero mucho más fácil e intuitivo de utilizar para el usuario.
- **Modo comando**: Se basa en la introducción de comandos mediante el teclado en una línea identificada por un texto. Este identificador se conoce como prompt del sistema operativo. Este modo de uso de un SO sigue siendo imprescindible para la realización de muchas tareas de administración de sistemas, como puede ser la solución de problemas de arranque o la ejecución de ciertos procesos como la desfragmentación o el chequeo de discos duros. Se utiliza especialmente en administración Linux. Los sistemas operativos actuales suelen presentarse en modo gráfico, pero tienen acceso al modo comando.

1 La filosofía en la que se basa el micronúcleo es que sólo las funciones absolutamente esenciales del núcleo del sistema operativo deben permanecer en el núcleo. Las aplicaciones y los servicios menos esenciales se construyen sobre el micronúcleo como subsistemas externos que interactúan con éste (sistemas de archivos, de ventanas, servicios de seguridad, etc.) y que se ejecutan en algunos casos como procesos de usuario. El microkernel sólo necesitaría contener la funcionalidad básica para crear y comunicar procesos

2 El shell ofrece al usuario una interfaz con el sistema operativo separando al usuario de los detalles y presentándole el sistema operativo como un simple conjunto de servicios. El shell acepta las órdenes del usuario o las sentencias de control de trabajos, las interpreta, crea y controla los procesos según sea necesario.

1.4. CLASIFICACION DE LOS SISTEMAS OPERATIVOS

POR LOS SERVICIOS OFRECIDOS:

- Por el número de usuarios:
  - **Monousuario:** Los sistemas operativos monousuario son aquéllos que no soportan más de un usuario a la vez, sin importar el número de procesadores que tenga la computadora o el número de procesos o tareas que el usuario pueda ejecutar en un mismo instante de tiempo. Todos los recursos del sistema estarán disponibles para el único usuario posible.
  - **Multiusuario:** Los sistemas operativos multiusuario son capaces de dar servicio a más de un usuario a la vez, ya sea por medio de varias terminales conectadas a la computadora o por medio de sesiones remotas en una red de comunicaciones. No importa el número de procesadores en la máquina ni el número de procesos que cada usuario puede ejecutar simultáneamente.
- Por el número de tareas:
  - **Monotarea o monoprogramación:** Los sistemas monotarea son aquellos que sólo permiten una tarea a la vez por usuario. Puede darse el caso de un sistema multiusuario y monotarea, en el cual se admiten varios usuarios al mismo tiempo pero cada uno de ellos puede estar haciendo solo una tarea a la vez
  - **Multitarea o multiprogramación:** Un sistema operativo multitarea es aquel que le permite al usuario estar ejecutando varios procesos al mismo tiempo. Por ejemplo, puede estar editando el código fuente de un programa durante su depuración mientras compila otro programa, a la vez que está recibiendo correo electrónico en un proceso en background (segundo plano). Estos procesos comparten tiempo de CPU hasta la finalización de cada uno de ellos. Es común encontrar en ellos interfaces gráficas orientadas al uso de menús y el ratón, lo cual permite un rápido intercambio entre las tareas para el usuario, mejorando su productividad.
- Por el número de procesadores que es capaz de usar:
  - **Monoproceso:** Un sistema operativo monoproceso o uniproceso es aquél que es capaz de manejar solamente un procesador de la computadora, de manera que si la computadora tuviese más de uno le sería inútil.
  - **Multiproceso:** Un sistema operativo multiproceso es capaz de manejar más de un procesador en el sistema, distribuyendo la carga de trabajo entre todos los procesadores que existan en el sistema
  - **Simétricos**: los procesos o partes de ellos (*threads*, hebras o hilos) son enviados indistintamente a cualquiera de los procesadores disponibles, teniendo, teóricamente, una mejor distribución y equilibrio en la carga de trabajo bajo este esquema.
  - **Asimétricos**: el sistema operativo selecciona a uno de los procesadores el cual jugará el papel de procesador maestro y servirá como pivote para distribuir la carga a los demás procesadores, que reciben el nombre de esclavos.

POR LA FORMA DE OFRECER LOS SERVICIOS:

- **Centralizados**: mainframes que se encargan de todo el procesamiento. Los usuarios se conectan a terminales que no procesan. Actualmente se siguen utilizando los sistemas centralizados (servidor Linux o Windows que ejecuta los servicios de terminal como Windows Terminal Services o Linux Terminal Services Project) pero los terminales ya sí procesan muchas tareas por sí mismos.
- **Distribuidos:** un sistema distribuido es una colección de computadoras independientes que aparecen ante los usuarios del sistema como una única computadora. Los sistemas operati-

vos distribuidos desempeñan las mismas funciones que un sistema operativo normal, pero con la diferencia de trabajar en un entorno distribuido. Su misión principal consiste en facilitar el acceso y la gestión de los recursos distribuidos en la red. En un sistema operativo distribuido los usuarios pueden acceder a recursos remotos de la misma manera en que lo hacen para los recursos locales. Permiten distribuir trabajos entre un conjunto de procesadores. Puede ser que este conjunto de procesadores esté en un equipo o en diferentes, lo cual es transparente para el usuario.

- **En red**. Varios ordenadores unidos a través de algún medio de comunicación, con el objetivo de compartir recursos e información. Cada ordenador tiene su propio sistema operativo.
- **De escritorio** o **S.O. cliente**: utilizados en equipos de sobremesa o portátiles.

POR SU DISPONIBILIDAD:

- **Propietarios**: Son propiedad intelectual de alguna empresa (Windows). Significa esto que se necesitan licencias de uso para ejecutarlos y no se tiene acceso a su código fuente, o si se tiene acceso, no se tiene derecho a modificarlo ni distribuirlo.
- **Libres**. Garantizan las cuatro libertades del software:
  1. Libertad de usar el programa con cualquier propósito.
  2. Libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a las necesidades que tuviera el usuario.
  3. Libertad de distribuir copias del programa, con lo que se puede ayudar a otros usuarios.
  4. Libertad de mejorar el programa y hacer públicas dichas mejoras a otros usuarios, de modo que toda la comunidad se beneficie de ello.

  El software libre es siempre de **código abierto** (libertades 2 y 4)

  El software libre no tiene por qué ser **software gratuito** ya que puede venderse una distribución con manuales, etc., en cuyo caso pasa a ser **software comercial**.

  El software gratuito (*freeware*) puede incluir el código fuente pero no tiene por qué ser libre a no ser que garantice el derecho de modificación y distribución de esas modificaciones.

  Software de **dominio público** es aquel que no requiere licencia pues todo el mundo puede usarlo pero no tiene por qué ser libre (puede no incluir el código fuente).

Investiga 2:

1. Identifica las características del SO de tu ordenador.
2. Nombra 2 sistemas operativos de cada tipo según los servicios ofrecidos.
3. Nombra un SO de cada tipo según la forma de ofrecer servicios.

2. TIPOS DE APLICACIONES Y TIPOS DE LICENCIA

En función del tipo de software las aplicaciones pueden ser:

a) **Gratuitas (*freeware*)** o **comerciales**.

b) **Libres** o **propietarias**. Libres se basan en la distribución del código fuente junto con el programa, así como en las cuatro premisas del software libre. Que una aplicación sea libre no implica que sea gratuita.

   El software propietario es aquel en que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones).

c) ***Opensource*** **(**código abierto) o **privativas** (código fuente no disponible o restringido).

Una **licencia** es un contrato mediante el cual un usuario recibe de una persona o empresa el derecho de uso de su software cumpliendo una serie de términos y condiciones establecidas dentro de sus cláusulas.

Algunos tipos de licencias del software propietario:

- **OEM**: La venta del software forma parte de un equipo nuevo
- **Retail**: El programa en este caso es enteramente del usuario que puede cederlo a terceros o venderlo.
- **Por volumen**: un número determinado de equipos pueden usar el mismo código de licencia. No se puede ceder a terceros.

El software libre está sujeto a su vez a una serie de licencias, cada una de ellas con sus respectivas normativas.

Investiga 3:

¿Qué implican los siguientes términos en cuanto a tipos de licencia?

GPL

BSD

Copyleft

Licencia de software libre

Licencia de software propietario

Freeware

Shareware

Opensource

EULA

Indica cuales de estos términos pueden referirse a un mismo producto y cuales son contradictorios.

3. LOS GESTORES DE ARRANQUE

Un **gestor de arranque** (en inglés *bootloader*) es un programa sencillo que no tiene la totalidad de las funcionalidades de un sistema operativo, y que está diseñado exclusivamente para preparar todo lo que necesita el sistema operativo para funcionar. Normalmente se utilizan los cargadores de arranque multietapas, en los que varios programas pequeños se suman los unos a los otros, hasta que el último de ellos carga el sistema operativo.

En los ordenadores modernos, el proceso de arranque comienza cuando la unidad central de procesamiento ejecuta los programas contenidos en una memoria ROM en una dirección predefinida. El microprocesador se configura para ejecutar este programa, sin ayuda externa, al encender el ordenador. Este programa buscará en las unidades de almacenamiento del ordenador, otro programa que a su vez cargará el sistema operativo.

a. Windows XP, Windows Server 2003: NTLDR, boot.ini, NTDETECT.COM
b. Windows Vista, Windows 7, Windows Server 2008: Bootmgr, Winload.exe, ntowkrnl.exe
c. Linux: lilo o grub

Investiga 4:

Nombra dos gestores de arranque no asociados a ningún sistema operativo

Trabajo voluntario:

Elabora una exposición sobre los gestores de arranque, incluyendo los arranques de Windows y Linux y algunos gestores comerciales.

4. GESTIÓN DEL PROCESADOR

4.1. LOS PROCESOS Y LA PLANIFICACIÓN

La mayoría de los procesadores dan soporte, al menos, para dos modos de ejecución. Estos son el modo usuario, el menos privilegiado, bajo el cual se ejecutan las aplicaciones del usuario, y el modo del sistema (o modo de control o del núcleo), el más privilegiado. Las instrucciones privilegiadas son aquellas que pueden potencialmente dañar a otros procesos, como por ejemplo desactivar interrupciones, la lectura o la modificación de los registros de control o aquellas instrucciones que se utilizan para gestionar la memoria; estas instrucciones sólo pueden ser llevadas a cabo por el núcleo del sistema.

La razón por la que se utilizan dos modos de ejecución radica en la necesidad de proteger al S.O. y a los datos importantes del mismo de injerencias de programas de usuario. En el modo núcleo, el software tiene el control completo del procesador y todas sus instrucciones, registros y memoria. Este nivel no es necesario, y por seguridad tampoco conveniente, para los programas de usuario.

4.2. GESTIÓN DE PROCESOS

Todos los sistemas operativos de multiprogramación están construidos en torno al concepto de proceso.

La misión principal del procesador es ejecutar las instrucciones que residen en memoria principal y que forman parte de un programa.

Desde el punto de vista del procesador, éste ejecuta instrucciones, dentro de un repertorio, según una secuencia dictada por los valores cambiantes de un registro: el contador de programa (PC) que no es más que una especie de puntero a las instrucciones. A lo largo del tiempo este valor puede apuntar a código de programas diferentes que son parte de diferentes aplicaciones. La ejecución de un programa individual se conoce como proceso o tarea. El comportamiento de un proceso individual viene determinado por el listado de la secuencia en que se ejecutan sus instrucciones. Dicho listado se llama traza del proceso.

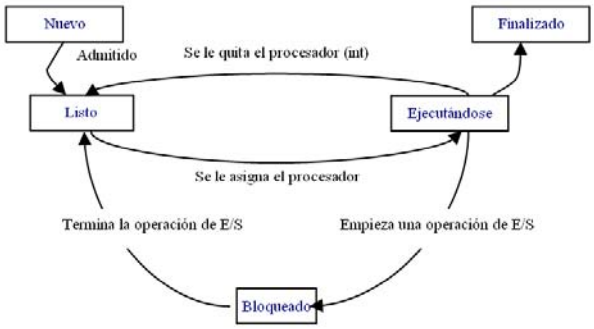
En cuanto a los procesos, los requisitos principales que debe satisfacer un sistema operativo son:

- El sistema operativo debe intercalar la ejecución de un conjunto de procesos para maximizar la utilización del procesador ofreciendo a la vez un tiempo de respuesta razonable.
- El sistema operativo debe asignar los recursos a los procesos en conformidad con una política específica (por ejemplo, ciertas funciones o aplicaciones son de prioridad más alta), evitando, al mismo tiempo el **interbloqueo**[3].
- El sistema operativo podría tener que dar soporte a la comunicación entre procesos y la creación de procesos por parte del usuario, labores que pueden ser de ayuda en la estructuración de las aplicaciones.

4.3. ESTADOS DE UN PROCESO

Un proceso cargado en memoria se puede encontrar, en un momento determinado, en tres estados posibles:

- Ejecución (usando el procesador)
- Listo (proceso que está preparado para ejecutarse en cuanto se le dé la oportunidad).
- Bloqueado (proceso que no se puede ejecutar hasta que no se dé un determinado suceso, como puede ser la terminación de una E/S). Normalmente su ejecución no puede continuar porque ha de esperar datos de entrada que todavía no están disponibles.



Se puede considerar que un proceso está formado por los tres componentes siguientes:

- Código ejecutable del programa
- Los datos asociados necesarios para ejecutar el programa (variables, pila, buffers, etc.).
- El contexto de ejecución del proceso: atributos usados por el S.O. para el control del proceso (contenido del registro PC indicando la próxima instrucción a ejecutar y del resto de registros de la CPU, memoria utilizada, archivos abiertos, etc.)

Periódicamente, el S.O. decide parar la ejecución de un proceso y arrancar la de otro, por ejemplo, porque el primero ha consumido su tiempo de procesador. Cuando se suspende temporalmente la ejecución de un proceso, debe reiniciarse posteriormente en el mismo estado en que se encontraba cuando se paró. Esto implica que toda la información referente al proceso debe almacenarse explícitamente en alguna parte durante su suspensión. Por ejemplo, si el proceso tuviera varios ficheros abiertos, se debería guardar en algún sitio la posición exacta en donde se encontraba el proceso en cada fichero, de forma que una instrucción de lectura emitida después de que el proceso rearrancase, leyera los datos apropiados.

El contexto de ejecución incluye toda la información que el sistema operativo necesita para administrar el proceso y que el procesador necesita para ejecutarlo correctamente. Esta información reside en el llamado bloque de control del proceso o descriptor de proceso. Diferentes sistemas organizarán esta información de diferente manera.

Veamos qué tipo de información podría usarse en el bloque de control de proceso en un sistema operativo, sin detallar la forma en que ésta se organizaría:

3 Un interbloqueo se produce básicamente si hay dos procesos que necesitan los mismos recursos para continuar y cada uno de ellos se ha apropiado de uno de los recursos. Cada proceso espera indefinidamente al recurso que le falta.

1. **Identificación del proceso**: Número identificador del proceso, del padre (si existe), del usuario, etc.
2. **Información del estado del procesador:** el contenido de los registros del procesador, que debe salvarse para poder restaurarse cuando el proceso reanude su ejecución.
3. **Información de Control del Proceso**: información adicional necesaria para que el S.O. controle y coordine los diferentes procesos activos: estado y prioridad de los procesos, dependencias padre-hijo, áreas de memoria asignadas, recursos en uso por cada proceso, etc.

Cuando se añade un proceso a los que ya está administrando el sistema operativo, éste debe crear las estructuras de datos que se utilizan para administrar un proceso y debe asignar el espacio de direcciones que va a utilizar (direcciones de memoria a las que puede acceder). Estas acciones constituyen la creación de un nuevo proceso.

4.4. CAMBIO DE PROCESO

Un cambio de proceso puede producirse en cualquier momento en que el sistema operativo haya tomado el control a partir del proceso que esté ejecutándose. Los sucesos posibles que pueden dar el control al S.O. son una interrupción, un cepo o una llamada al supervisor. En primer lugar, vamos a tener en cuenta las interrupciones del sistema: una conocida simplemente como interrupción y otra como cepo.

Una interrupción es un mecanismo generado por algún tipo de suceso que es externo e independiente del proceso que está ejecutándose (como puede ser la culminación de una operación de E/S).

Un cepo tiene que ver con una condición de error o de excepción generada dentro del proceso que está ejecutándose (como puede ser un acceso ilegal a un archivo).

Una llamada al supervisor se produce cuando un proceso quiere realizar una operación para la que no tiene permisos, entonces cede el control a un módulo del SO.

En una interrupción ordinaria el control se transfiere primero al gestor de interrupciones quien lleva a cabo unas tareas básicas y, después, se salta a una rutina del sistema operativo que se ocupa del tipo de interrupción que se ha producido. Algunos ejemplos son los siguientes:

- **Interrupción de reloj**: El sistema operativo determina si el proceso que está en ejecución ha estado ejecutándose durante la fracción máxima de tiempo permitida. Si esto ocurre, el proceso debe pasar al estado listo y se debe pasar el control a otro proceso (ver Algoritmos de Planificación)
- **Interrupción de E/S**: Cuando el sistema operativo determina que se ha producido una operación de E/S y ésta constituye un suceso que han estado esperando uno o más procesos, modifica el estado de éstos de bloqueado a listo. Entonces deberá decidir si se reanuda la ejecución del proceso que está ejecutándose actualmente o si se le expulsa en favor de otro de mayor prioridad.
- **Fallo en memoria**: El procesador encuentra una referencia a una dirección de memoria virtual de una palabra que no se halla en memoria interna. El sistema operativo debe traer el bloque que contiene la referencia de la memoria secundaria a la principal. Después de hacer la solicitud para extraer el bloque de memoria secundaria el proceso que causó el fallo de memoria se pasa a estado bloqueado. Después de que el bloque en cuestión se cargue en memoria dicho proceso pasará al estado de listo.

En los cepos el sistema operativo determina si el error es fatal. En ese caso, el proceso que se estaba ejecutando se termina y se produce un cambio de proceso. Si por el contrario no se trata de un error fatal, la acción del sistema operativo dependerá de la naturaleza del mismo y del diseño del sistema operativo. Se puede intentar algún procedimiento de recuperación o, simplemente, reanudar el mismo proceso que estaba ejecutándose.

4.5. PLANIFICACIÓN DEL PROCESADOR

El planificador de procesos es un elemento fundamental en los sistemas operativos multitarea. Se encarga de indicar qué proceso debe ejecutarse en cada momento y cuándo un proceso puede pasar de un estado a otro, según se dé un determinado suceso, como acabamos de ver.

El conjunto de criterios o reglas que sigue el planificador para decidir en cada momento qué proceso debe ejecutarse es lo que se denomina algoritmo de planificación.

El problema que plantea la planificación de procesos es que cada uno de ellos es único e impredecible. Algunos pasan mucho tiempo esperando operaciones de E/S y otros, aquellos con pocas o ninguna operación de este tipo, pasarían horas ocupando el procesador. El planificador no sabe, en principio, cuanto tiempo pasará hasta que un proceso termine o pase al estado de "bloqueado". Para controlar el tiempo que lleva en ejecución, activa una especie de cronómetro (un temporizador hardware, o reloj) que indica el tiempo de procesador consumido y produce interrupciones periódicas. Con cada interrupción de reloj, el sistema operativo recupera el control y decide si debe dejar que el proceso actual continúe ejecutándose, o si, por el contrario, éste ya ha utilizado el procesador durante un tiempo suficientemente largo y, por tanto, es el momento de suspender su ejecución y ceder el uso del procesador a otro proceso.

TIEMPO COMPARTIDO

El concepto de tiempo compartido (*Time sharing*) hace referencia a compartir un recurso computacional entre muchos usuarios. Al permitir que un gran número de usuarios interactúe concurrentemente en una sola computadora, el coste del servicio de computación baja drásticamente, mientras que al mismo tiempo hace la experiencia computacional mucho más interactiva ya que cada usuario obtiene del ordenador un rendimiento casi igual al que obtendría si fuese el único en usarlo. De hecho, el usuario percibe la sensación de ser el único. Para llevar esto a cabo hace falta un algoritmo de planificación.

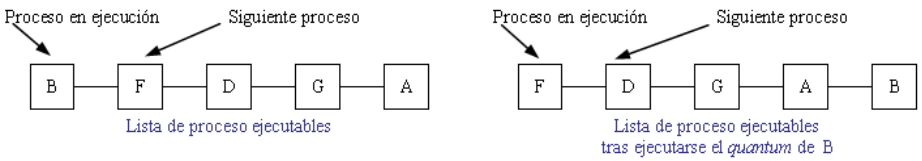
4.6. ALGORITMOS DE PLANIFICACIÓN

MULTIPROGRAMACIÓN CLÁSICA

Bajo esta técnica se cambia de un proceso a otro cuando el que está ejecutándose entra en una operación de E/S (o termina). Esto hace que procesos con mucho tiempo de cálculo/operaciones lógicas y pocas operaciones de E/S monopolicen la CPU.

PLANIFICACIÓN POR TORNEO, TIEMPO PARALELO O TURNO ROTATORIO.

Consiste en crear una lista con los procesos pendientes de ejecutarse. A cada uno de ellos se le asigna un intervalo de tiempo fijo o periodo T, conocido como *quantum* o cuanto (usualmente del orden de décimas de segundo). Cuando un proceso finaliza su *quantum* el planificador pasa el control al siguiente programa y así sucesivamente. Cuando se acaba el *quantum* de un proceso, éste pasa al final de la lista.



Este sistema utiliza un circuito contador activado por el reloj del sistema (el cronómetro del que hablamos antes) que, cuando su salida llega a una determinada cuenta (ha transcurrido un tiempo de ejecución para un proceso igual al *quantum* que tenía asignado) provoca una interrupción de la CPU que es atendida por un módulo gestor de interrupciones y da el turno al siguiente trabajo a través

del planificador. Si antes de que expire el *quantum* el proceso se bloquea (o ha terminado), el planificador da paso al siguiente proceso de la lista en el momento del bloqueo.

PLANIFICACIÓN POR PRIORIDAD.

A cada proceso se le asigna una prioridad de forma que siempre se concede el procesador al proceso ejecutable de más alta prioridad. Para evitar que los procesos de alta prioridad se ejecuten por tiempo indefinido, el planificador puede decrementar la prioridad del proceso que está ejecutándose cada cierto tiempo. Cuando su prioridad llegue a ser menor que la del proceso que le sigue, se realizará un cambio de proceso**.**

LISTA DE ESPERA CON INTERVALOS MÚLTIPLES.

Este algoritmo intenta beneficiar a los procesos que necesitan utilizar mucho el procesador. Para ello asigna intervalos de tiempo de ejecución diferentes según el proceso. El proceso empieza a ejecutarse con un *quantum* normal (igual para todos). Cuando ha terminado su *quantum* sin ser bloqueado, se incrementa éste para la siguiente vez. Cuando un proceso entra en estado "bloqueado", por ejemplo por la ejecución de una operación de E/S, vuelve a comenzar con el *quantum* inicial. Se debe establecer, no obstante, un *quantum* máximo para evitar que un proceso vaya aumentando el suyo hasta acaparar el procesador, paralizando el resto de los procesos.

PRIORIDAD AL MÁS CORTO.

Especialmente indicado para sistemas por lotes en los que se conoce de antemano el tiempo de ejecución.

Ejemplo:

Tenemos 3 procesos (P1, P2 y P3) que se lanzan simultáneamente en tiempo=0.

- P1 necesita un total de 25 ms de microprocesador para finalizar, siendo de cálculo los primeros 15 ms y de E/S los otros 10 ms.
- P2 lee desde disco durante 3 ms, procesa durante 1 ms e imprime durante 6 ms. Así hasta 3 ciclos.
- P3 es sólo de cálculo y necesita 15 ms.

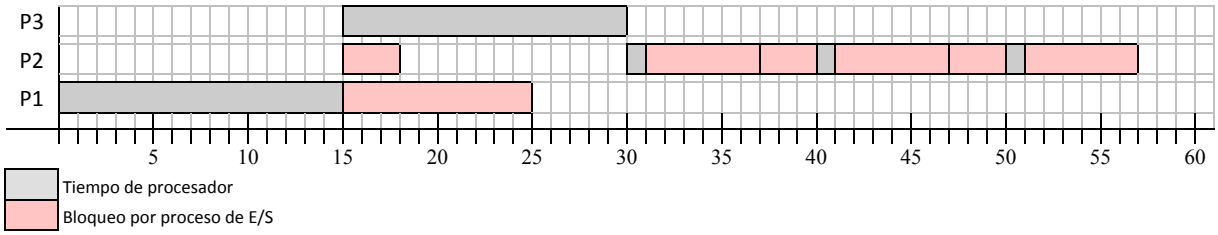
Calcular cuánto tiempo necesita en total para ejecutar los 3 procesos usando monoprogramación, multiprogramación clásica, multiprogramación en tiempo paralelo con quantum de10 ms y multiprogramación en tiempo paralelo con quantum de 5 ms.

Nota: en este ejemplo, y en los ejercicios, no consideraremos ni reflejaremos los tiempos en los que el sistema operativo toma el control del procesador, bien sea para realizar los cambios de proceso y/o para la gestión de interrupciones. Es decir, reflejaremos sólo el tiempo que el procesador invierte en ejecutar los procesos propiamente dichos.

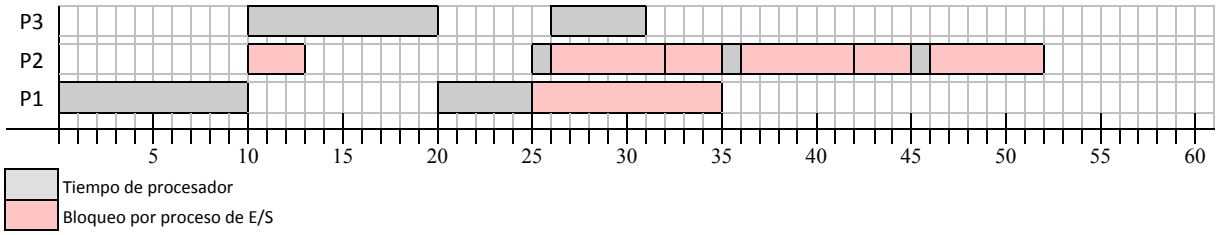
Solución:

a) monoprogramación: 25+[(3+1+6)*3]+15 = 70 ms

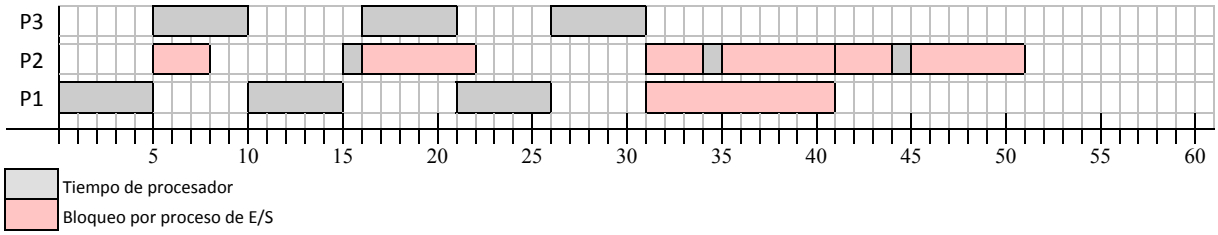
b) multiprogramación clásica: 57ms



c) multiprogramación tiempo paralelo con un quantum de 10 ms: 52 ms



d) multiprogramación tiempo paralelo con un quantum de 5 ms: 51 ms



Ejercicios:

Realiza los ejercicios relacionados en "[2a] Ejercicios de planificación procesos"

5. GESTIÓN DE LA MEMORIA

En un sistema monoprogramado, la memoria principal se divide en dos partes: una para el sistema operativo y otra para el programa de usuario que se ejecuta en ese instante.

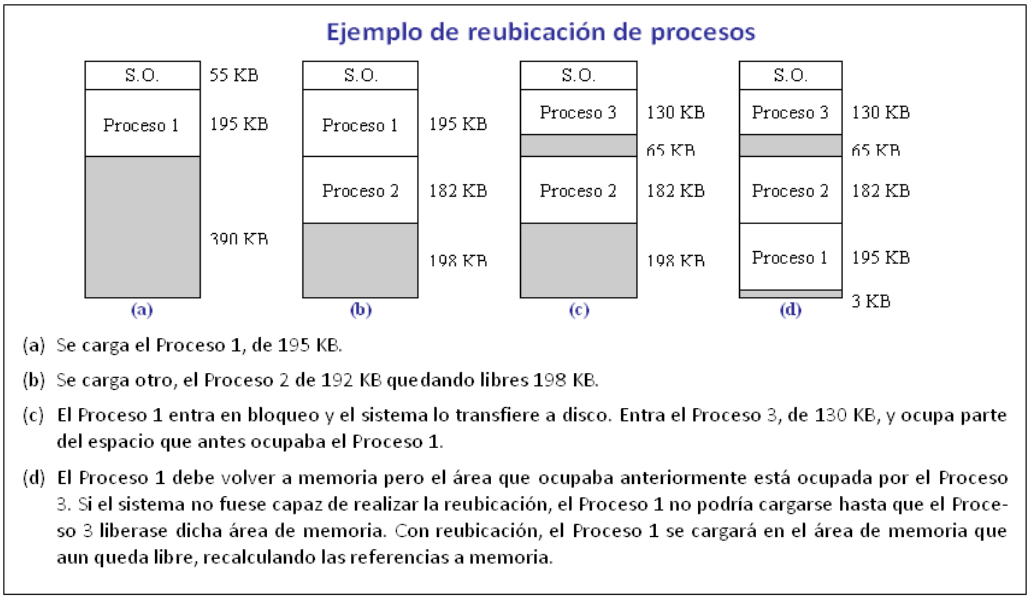
En un sistema multiprogramación, la parte de "usuario" de la memoria debe subdividirse en más partes para dar cabida a varios procesos. La tarea de subdivisión la lleva a cabo dinámicamente el sistema operativo y se conoce como gestión de la memoria.

Se entiende que una gestión de la memoria efectiva es esencial en un sistema multiprogramado. Si sólo hay unos pocos procesos en memoria, la mayor parte del tiempo estarán esperando concluir sus operaciones de E/S y es posible que se produzcan tiempos muertos de CPU. Por ello, hace falta repartir eficientemente la memoria para albergar tantos procesos como sea posible.

REUBICACIÓN.

Es imposible que el programador conozca, de antemano, qué otros programas residirán en memoria en el momento en que se ejecute el suyo. El sistema operativo debe poder cargar y descargar los procesos en la memoria principal durante su tiempo de ejecución. Con esto se consigue maximizar el uso del procesador (más procesos listos para su ejecución) y una utilización más eficiente del

espacio. Esto lo consigue mediante técnicas de intercambiabilidad memoria principal-disco (*swapping*). Para poder llevar a cabo estas funciones se requiere espacio para intercambio en almacenamiento secundario que puede ser una partición de disco o un fichero. El sistema operativo debe saber dónde situar un programa en concreto en la memoria y poder ubicarlo en zonas distintas si fuese necesario a resultas del mecanismo de *swapping*. Además en todo momento debe conocer la ubicación de la información de control del proceso.



(a) Se carga el Proceso 1, de 195 KB.

(b) Se carga otro, el Proceso 2 de 192 KB quedando libres 198 KB.

(c) El Proceso 1 entra en bloqueo y el sistema lo transfiere a disco. Entra el Proceso 3, de 130 KB, y ocupa parte del espacio que antes ocupaba el Proceso 1.

(d) El Proceso 1 debe volver a memoria pero el área que ocupaba anteriormente está ocupada por el Proceso 3. Si el sistema no fuese capaz de realizar la reubicación, el Proceso 1 no podría cargarse hasta que el Proceso 3 liberase dicha área de memoria. Con reubicación, el Proceso 1 se cargará en el área de memoria que aun queda libre, recalculando las referencias a memoria.

Además, el hardware del procesador y el software del sistema operativo deben ser capaces de traducir las referencias relativas a memoria encontradas en el código del programa a las direcciones físicas reales que reflejen la posición actual del programa en memoria principal.

PROTECCIÓN.

Cada proceso debe protegerse contra interferencias no deseadas de otros procesos, tanto accidentales como intencionadas. Así pues, el código de un proceso no puede hacer referencia a posiciones de memoria de otros procesos, con fines de lectura o escritura, sin permiso. La característica de reubicación de los procesos, sin duda, dificulta la protección de los mismos ya que se desconoce la ubicación de un programa en memoria principal y por tanto es imposible comprobar las direcciones absolutas durante la compilación para asegurar la protección. Por tanto, todas las referencias a memoria generadas por un proceso deben ser comprobadas en tiempo de ejecución para asegurarse que sólo hacen referencia a su zona de memoria.

No obstante, las exigencias de protección de memoria pueden ser satisfechas por el procesador (hardware) en vez de por el sistema operativo (software). Esto es debido a que el sistema operativo no puede anticiparse a todas las referencias a memoria que hará un programa. Incluso si esto fuera factible, sería prohibitivo en términos de tiempo consumido.

COMPARTICIÓN.

Cualquier mecanismo de protección debe tener la flexibilidad de permitir el acceso de varios procesos a una misma zona de memoria principal compartida. Por ejemplo, si una serie de procesos están ejecutando el mismo programa sería beneficioso permitir que cada proceso accediese a la misma copia del programa en lugar de tener una por cada uno de ellos.

5.1. TÉCNICAS DE GESTIÓN DE MEMORIA

5.1.1 PARTICIONES ESTÁTICAS

Es la forma más sencilla de organizar la memoria para poder tener varios procesos simultáneamente residentes en memoria. Consiste en dividir la memoria en n particiones, quizás de tamaño diferente, cada una de las cuales contendrá un programa. Las direcciones base[4] son las direcciones de comienzo de cada partición. El tamaño de cada partición es determinado por el operador o por el sistema operativo por lo que el tamaño de las particiones ya se sabe antes de cargar los programas en memoria. Normalmente, el número de particiones y su tamaño no puede cambiarse una vez establecido en una sesión de trabajo por lo que debería reiniciar el sistema para cambiarlas.

Algoritmo de ubicación.

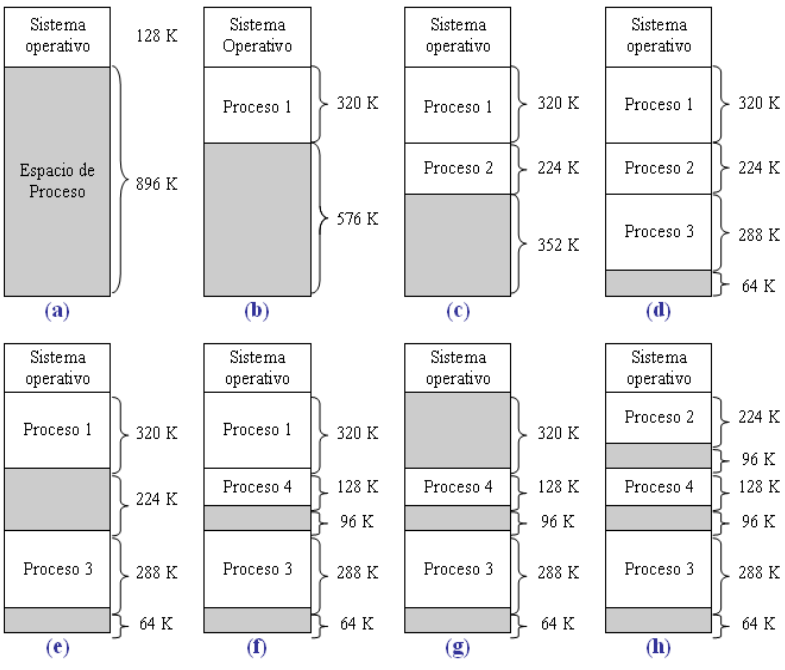
El sistema operativo mantiene una tabla en la que cada fila corresponde a una partición, conteniendo la posición base de la misma, su tamaño (no todas las particiones tienen por qué ser iguales) y el estado de la partición (ocupada o libre). El planificador de procesos, una vez que una partición queda libre, hace que se introduzca el programa de máxima prioridad que quepa en ella.

Este método está hoy ampliamente superado y presenta básicamente dos problemas:

- Un programa puede no caber en una partición, lo que obliga al programador a estructurar su aplicación en módulos que sí quepan.
- Efectúa un uso ineficiente de la memoria ya que cualquier programa, por pequeño que sea, ocupa una partición entera lo que provoca una fragmentación "interna": se malgasta el espacio interno de una partición.

5.1.2 PARTICIONES DINÁMICAS

Las particiones son variables en número y longitud. Conforme se carga un proceso se le asigna la memoria que necesita y no más. En la siguiente figura se muestra un ejemplo que usa 1MB de memoria principal. Al principio, la memoria principal está vacía, exceptuando el sistema operativo.

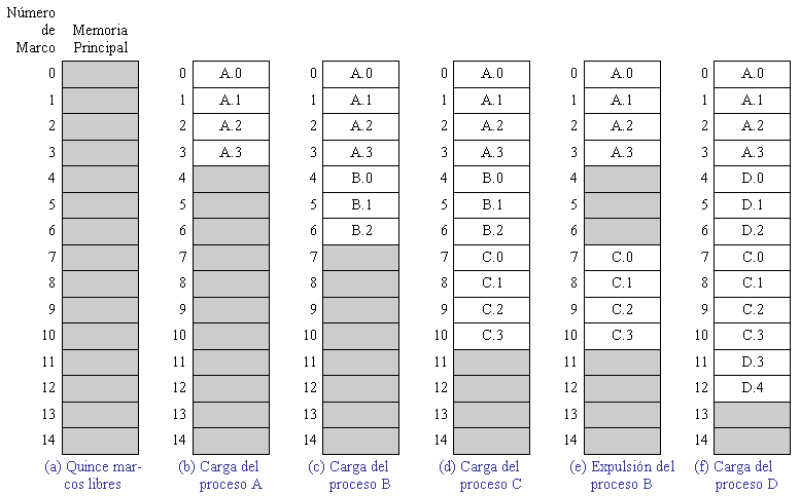


Efectos de la partición dinámica

Como se ve, este método comienza bien, pero finalmente, desemboca en una situación en la que hay un gran número de huecos pequeños en memoria, problema que puede resolverse mediante la compactación de los huecos libres. La compactación se puede efectuar cambiando de lugar los programas en ejecución aprovechando la posibilidad de reubicarles que debe ofrecer el sistema operativo.

[4] Dirección donde se carga la primera instrucción del proceso y, a partir de la cual, se calculan el resto de las direcciones que utiliza el proceso

5.1.3 PAGINACIÓN



Asignación de páginas de procesos a marcos libres

Con este procedimiento, la memoria principal se estructura en trozos de tamaño fijo llamados marcos o marcos de página o simplemente bloques. Cada proceso está dividido también en pequeños trozos de tamaño fijo y del mismo tamaño que los marcos de memoria llamados páginas. De esta forma, las páginas pueden asignarse a marcos de página libres.

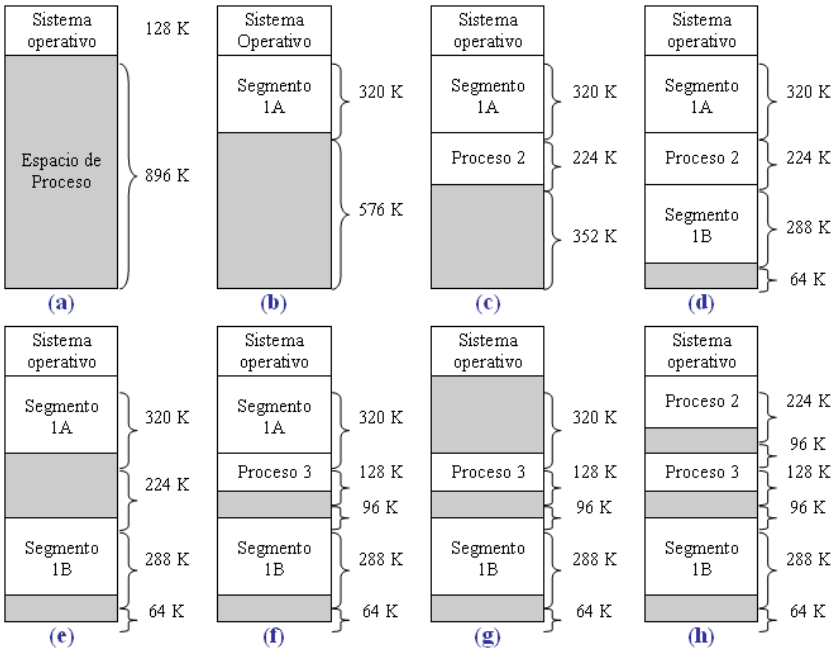
El fundamento de la paginación reside en que no es necesario que un programa se almacene en posiciones consecutivas de memoria. Las páginas se almacenan en marcos libres, independientemente de que estén o no contiguos. Eso sí, cada bloque contendrá, obviamente, instrucciones consecutivas.

Una instrucción de programa se localiza dando el marco de memoria y la dirección relativa dentro del bloque.

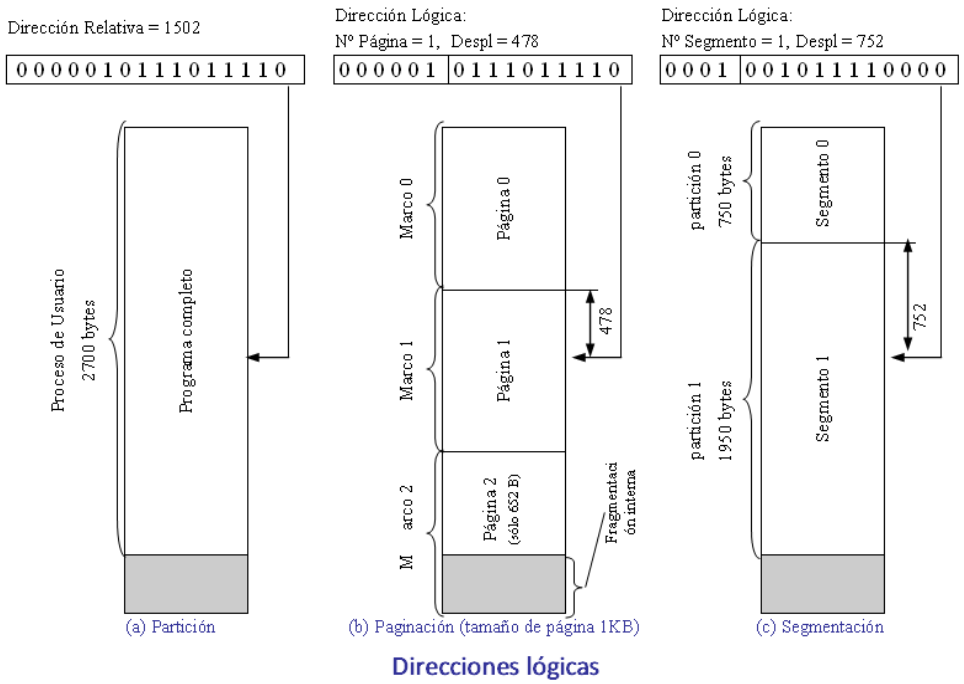
5.1.4 SEGMENTACIÓN

El programa se considera divido en segmentos. Hay que tener en cuenta que nos referimos a los segmentos, en este caso, como a grupos lógicos de información (código, pila, datos, etc.) definibles por el programador o asignados por el compilador, de forma que se puede decir que un programa es una colección de segmentos. Como en paginación, una dirección lógica segmentada consta de dos partes, en este caso un número de segmento y un desplazamiento.

La gestión la realiza el sistema operativo como en el caso de las particiones dinámicas, sólo que cada partición no corresponde a un programa sino a un segmento del mismo y las particiones que ocupa un programa no tienen por qué estar contiguas.



Reubicación de la segmentación



5.2. MEMORIA VIRTUAL

La memoria virtual permite a los usuarios hacer programas de una capacidad superior a la que físicamente tiene la computadora. Los dos enfoques básicos de memoria virtual son la paginación y la segmentación, aunque también se pueden combinar ambos enfoques en un único esquema de gestión de memoria. La memoria virtual exige un soporte tanto de hardware como de software. El soporte hardware lo proporciona el procesador e incluye la traducción dinámica de direcciones virtuales a direcciones físicas y la generación de interrupciones cuando una página o segmento referenciado no están en memoria principal. Estas interrupciones activan el software de gestión de memoria del sistema operativo.

Primero vamos a centrarnos en los aspectos del hardware de la memoria virtual, considerando el uso de la paginación, la segmentación o una combinación de ambas. Tanto una como otra técnica presentan dos características:

1.- Todas las referencias a memoria dentro de un proceso se hacen por medio de direcciones lógicas que se traducen dinámicamente en direcciones físicas durante la ejecución. Esto facilita la reubicación de un proceso si fuese necesario.

2.- Un proceso puede dividirse en partes, páginas o segmentos, y no es necesario que estas partes se encuentren consecutivas en la memoria principal durante la ejecución. Esto es posible por la combinación de la traducción dinámica de direcciones en tiempo de ejecución y el uso de una tabla de páginas o de segmentos.

La consecuencia inmediata de estas ventajas es que no es necesario, por tanto, que todas las páginas o todos los segmentos de un proceso estén en memoria durante la ejecución. Basta con que estén los segmentos o páginas donde se encuentren las siguientes instrucciones a ejecutar o la siguiente zona de datos a la que se tiene que acceder.

Esto puede suponer mejoras en la utilidad del sistema puesto que:

1.- Se pueden conservar más procesos en memoria puesto que hay más espacio disponible. Esto implica un mayor aprovechamiento del procesador al haber menos probabilidad de que todos los procesos en ejecución estén en un momento determinado en estado “bloqueado” (esperando una E/S).

2.- Es posible ejecutar procesos muy grandes e, incluso, un único proceso puede superar la capacidad de la memoria ya que el sistema operativo sólo cargará en memoria interna los fragmentos que necesite de él en un momento determinado, permaneciendo el resto en memoria secundaria hasta ser requeridos.

Como los procesos se ejecutan sólo en memoria principal, a esta memoria se le denomina memoria real. Pero un programador o un usuario percibe en potencia una memoria mucho mayor que está situada en el disco, es decir, podemos hablar de una memoria virtual.

El fin que se persigue es que en un instante dado, en memoria sólo se tengan unos pocos fragmentos de un proceso y, por tanto, poder mantener más procesos en memoria. Es el sistema operativo el encargado de mantener este esquema. Pero esto puede hacer necesario el tener que evacuar un fragmento para hacerle sitio a otro que se necesite para la ejecución. Además, si se expulsa un fragmento que seguidamente va a ser utilizado, obligatoriamente deberá el sistema volver a cargarlo casi de forma inmediata. En definitiva, el sistema implica un trasvase de fragmentos desde memoria secundaria hasta memoria principal. Esto podría suponer un hándicap que hiciera ineficaz esta gestión de la memoria, puesto que demasiadas operaciones de este tipo pueden conducir a lo que se conoce como *trashing* (hiperpaginación), es decir, que el procesador consuma más tiempo en el intercambio de fragmentos que ejecutando instrucciones.

Ejercicios:

Realiza los ejercicios relacionados en "[3] Problemas de memoria virtual"

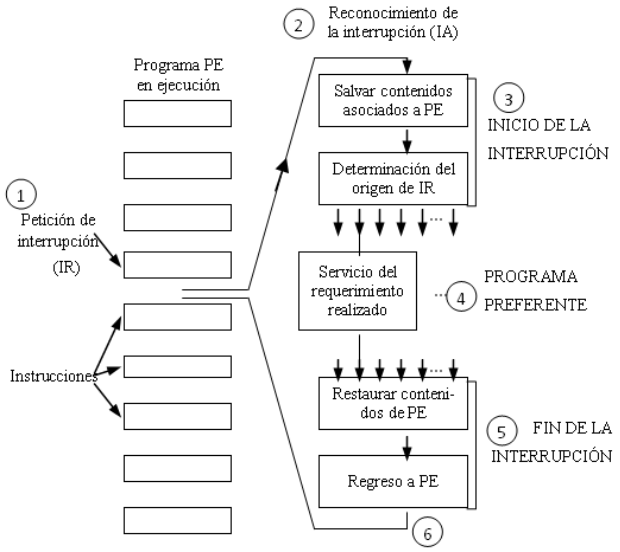
6. GESTIÓN DE LAS INTERRUPCIONES.

Las interrupciones son un método del que disponen los elementos hardware (E/S, memoria, reloj) e incluso los procesos para hacer notar a la CPU la aparición de alguna circunstancia que requiera su intervención.

Se utilizan las interrupciones generalmente por dos motivos: permitir una comunicación sin bloqueo con los periféricos externos y conmutar las tareas dentro de un planificador de procesos (como hemos visto en el apartado correspondiente).

Las interrupciones de E/S son generadas por un controlador de E/S para indicar que una operación ha terminado normalmente o para indicar diversas condiciones de error.

El acontecimiento de una interrupción desencadena una serie de sucesos, tanto en hardware como en el software del sistema.



Esquema que muestra la gestión de una interrupción de un programa PE para atender a otro

1.- Petición o demanda de interrupción (interruption request): La petición se realiza por medio de una señal eléctrica (señal de estado) enviada a la CPU.

2.- Por lo general, la CPU no atiende inmediatamente la petición de la interrupción, sino que antes acaba de ejecutar la instrucción que se halla en curso. Por medio de instrucciones adecuadas, las CPU actuales pueden inhibir las peticiones de interrupción cuando se ejecuta un módulo o programa de alto privilegio, como el núcleo del sistema operativo, por ejemplo, y posteriormente habilitar o atender las peticiones que mientras tanto se hubiesen producido. También hay técnicas de arbitraje de peticiones de interrupción, de forma que cuando hay varias peticiones pendientes se prevé cuál atender primero.

3.- Se atiende la interrupción (interruption acknowledge) poniéndose en marcha una rutina o programa de servicio de inicio de interrupción que analiza la causa de la interrupción, salva el contenido de los acumuladores/registros de la CPU (contador de programa, PSW, etc.) y pasa el control al programa preferente. La salvaguarda de estos elementos se hace para que no se pierdan con la ejecución del programa preferente.

4.- Se ejecuta la rutina o programa preferente, que atiende el requerimiento de la interrupción.

5.- Se ejecuta la rutina de fin de servicio de interrupción que restaura el contenido de los registros de la CPU con los datos del programa interrumpido.

6.- La rutina de fin de servicio de interrupción pasa el control de ejecución al programa interrumpido, continuándose éste.

Hay que tener en cuenta que un programa que interrumpe a otro puede ser, a su vez, interrumpido y así sucesivamente.

7. GESTIÓN DE LA E/S

La gestión de la E/S es uno de los aspectos más confusos en el diseño de los SO. Las características de los dispositivos de E/S suelen diferir mucho de las de la CPU, por ejemplo en la velocidad, ancho de palabra, etc. Aparte de esto, la gran variedad de dispositivos impide construir una solución general para la gestión de los mismos.

Normalmente las unidades de E/S constan de dos elementos: mecánico, que es el dispositivo propiamente dicho y electrónico al que se le suele denominar controlador. El sistema operativo casi siempre trata con el controlador.

Todos los controladores disponen de unos cuantos registros para comunicar con el procesador. Estos registros pueden formar parte del espacio normal de direcciones de memoria (tienen un área de memoria reservada para esta función). En otros casos se utiliza un espacio de direcciones especial para E/S, con una parte del mismo asignada a cada controlador. Para realizar operaciones de E/S, el SO escribe comandos en los registros del controlador, el controlador acepta el comando y mientras el procesador puede dejarlo que haga su trabajo solo y ocuparse entretanto de otras cosas.

7.1. TÉCNICAS DE E/S

E/S PROGRAMADA

Con esta técnica, ya obsoleta, el módulo de E/S no avisa al microprocesador de nada en particular, es decir, no lo interrumpe. Es el microprocesador el responsable de comprobar periódicamente el estado del módulo de E/S hasta saber que se ha completado una operación y cual es el resultado (comprobar por ejemplo si ha habido un error).

El procesador es también responsable de extraer los datos de la RAM cuando se va a hacer una salida y de cargarlos en ella en el caso de una entrada. El software está diseñado de tal forma que se le otorga al microprocesador el control directo sobre la operación de E/S incluyendo la comprobación del estado del dispositivo, el envío de órdenes de lectura/escritura y la transferencia de los datos. Se trata de un proceso que consume tiempo y mantiene ocupado al procesador de forma innecesaria.

E/S DIRIGIDA POR INTERRUPCIONES

Con esta técnica el microprocesador envía una orden de E/S al módulo correspondiente y mientras se dedica a hacer otra cosa, sin tener que comprobar el estado del módulo de E/S cada cierto tiempo. El módulo de E/S interrumpirá al microprocesador cuando necesite intercambiar información con él. El microprocesador obtiene los resultados de la operación y el estado del dispositivo leyendo uno o más bytes de información de los registros del controlador mencionados anteriormente.

Se sigue consumiendo tiempo de microprocesador debido a que cada palabra de datos que va de la memoria al módulo de E/S o viceversa debe pasar por él.

DMA

Muchos controladores tienen acceso directo a memoria (DMA, *Direct Memory Access*), útil sobre todo cuando se tienen que mover grandes cantidades de información.

La técnica funciona de la siguiente forma: cuando el microprocesador desea leer o escribir un bloque de datos emite una orden para el módulo DMA y le envía la información siguiente:

- Si lo que solicita es una lectura o escritura
- La dirección del dispositivo de E/S involucrado
- La dirección inicial de memoria a partir de la cual se va a leer o escribir
- El número de palabras a leer o escribir

Seguidamente el microprocesador continúa con otro trabajo y mientras el módulo DMA se ocupa de la E/S, transmitiendo la información desde o hacia la memoria sin pasar por el microprocesador. Cuando se completa la transferencia, el módulo DMA envía una señal de interrupción al microprocesador. De esta forma el microprocesador se ve involucrado en la E/S sólo al principio y al final de la operación.

Si no existiera DMA, para leer datos de un disco por ejemplo, el controlador tendría que leer el bloque de la unidad de disco y llevarlo al buffer interno del controlador. Después generaría una interrupción. Cuando el sistema operativo ejecutara la rutina de interrupción, podría leer el bloque de disco del buffer del controlador palabra a palabra y escribirla en la memoria. Cada palabra leída del controlador necesitaría tiempo de procesador. El DMA se inventó para liberar al procesador de esta actividad de bajo nivel.

7.2. ALMACENAMIENTO INTERMEDIO DE LA E/S: *BUFFERING*

Para evitar las consecuencias que se derivan de la diferencia en la velocidad de funcionamiento de los dispositivos y la CPU se suele utilizar un buffer. Un **buffer** es un almacén de información. El buffer del controlador se utiliza para guardar temporalmente los datos implicados en una operación de E/S. Por ejemplo, si se quiere escribir en una impresora, se carga la información a escribir desde memoria principal al buffer. Posteriormente, el controlador mandará dicha información desde el buffer a la impresora. Si se trata de una unidad de entrada, la transferencia de información se puede hacer por adelantado de forma que la CPU tenga los datos en el buffer cuando los necesite.

El almacenamiento intermedio es una técnica que soluciona los problemas de “horas punta” en la demanda de E/S. Sin embargo, no existe un tamaño de los buffers que asegure a un dispositivo de E/S ir al mismo ritmo que un proceso cuando la demanda media del proceso es mayor que la que el dispositivo puede admitir. Incluso si se dispone de varios buffers, al final todos se llenarán y el proceso tendrá que quedarse esperando tras operar con una determinada cantidad de datos. Sin embargo, en un entorno de multiprogramación, con la variedad de actividades de E/S y de procesamiento que hay que realizar, el almacenamiento intermedio es una herramienta que puede incrementar la eficiencia del sistema operativo y el rendimiento de los procesos individuales.

8. LENGUAJES DE PROGRAMACIÓN

Se considera lenguaje de programación cualquier lenguaje artificial que puede utilizarse para definir una secuencia de instrucciones (programa) para su procesamiento por un ordenador. Existen dos niveles básicos de lenguajes, dependiendo del nivel de acercamiento al lenguaje humano (normalmente, el inglés):

8.1. LENGUAJES DE BAJO NIVEL.

Son aquellos en los que las instrucciones están codificadas en un lenguaje más próximo a la máquina que al hombre. Los programas en lenguajes de bajo nivel, al ser órdenes directamente inteligibles por el procesador, son muy rápidos pero el lenguaje en sí es, a menudo, difícil de aprender.

Además, un programa en lenguaje de bajo nivel raramente puede ser ejecutado en otra máquina, ya que las órdenes son específicas de un procesador en concreto.

8.1.1 CÓDIGO MÁQUINA.

Son aquellos en los que las órdenes se codifican de forma directamente entendible por el ordenador, es decir, en binario. Tanto las ordenes como los datos deben introducirse en binario. Además, las órdenes deben corresponderse con aquellas que comprende directamente el procesador, por lo que una orden simple, como sumar dos números almacenados en memoria, debe descomponerse en todos sus pasos:

leer la dirección correspondiente al primer dato,

llevarlo al circuito sumador,

leer el segundo dato,

llevarlo al circuito sumador,

asignar una dirección de memoria para el nuevo dato,

trasladar el dato resultante del registro temporal del procesador a memoria.

Todo esto codificado a base de 0 y 1.

Al no necesitar traducirse, son muy rápidos en la ejecución aunque el tiempo necesario para escribir el programa y depurarlo es mucho mayor que en otros lenguajes.

Hoy en día este tipo de lenguajes está en desuso aunque era el único que entendían los primeros ordenadores, como el ENIAC.

8.1.2 LENGUAJE ENSAMBLADOR.

Segundo paso en el desarrollo de los lenguajes. Funcionan prácticamente igual que el código máquina, salvo que se sustituye el binario en las órdenes por código alfanumérico mnemotécnico (generalmente unas pocas letras) y los datos por hexadecimal. Además, algunas operaciones básicas se reagrupan en un sólo código por lo que la orden anterior quedaría resumida a:

```
CAR A, M(H0007)
SUM A, M(H0009)
MEM M(H000C), A
```

Este lenguaje simplifica claramente el proceso de escritura y depuración de los programas, ya que es más fácil aprenderse los códigos de las instrucciones y, al estar los datos en hexadecimal, también resulta más fácil la conversión a decimal.

Sin embargo, presenta el mismo problema que el código máquina en cuanto a su utilización en diversos ordenadores ya que los lenguajes ensambladores son específicos de cada procesador e incompatibles entre si.

Además, es necesario traducir el programa generado por el usuario (código o programa fuente) a código máquina. Esto se hace con los ensambladores, que generan a partir del fuente el programa ejecutable.

Este programa ejecutable no necesita ser ensamblado cada vez que se ejecuta pero si cada vez que realicemos cambios en el fuente.

8.2. LENGUAJES DE ALTO NIVEL.

Son aquellos en los que el programa fuente se aproxima más al lenguaje humano que al código máquina. Entre sus características podemos destacar:

- Portabilidad: Son independientes del hardware, lo que permite al programador abstraer el problema del entorno en que será utilizado posteriormente. Un programa puede generarse en un tipo de máquina y ejecutarse posteriormente en otras de distinta arquitectura.
- Utilizan ordenes englobantes (una sola orden puede equivaler a cientos de ordenes máquina). Siguiendo el ejemplo anterior, la única orden necesaria es `A = B + C`.
- Precisan ser traducidos a código máquina antes de ejecutarlos. Para ello se puede recurrir a la interpretación o a la compilación que explicaremos más adelante.
- Sus ordenes, aunque codificadas, suelen utilizar palabras claves, en inglés normalmente, (SORT ordenar) o acronimos (CLS CLear Screen)
- Los programas en lenguaje de alto nivel (módulo fuente) requieren ser traducidos a código ejecutable mediante compiladores (con los que obtenemos un módulo ejecutable) o interpretes.

El primer compilador para lenguajes de alto nivel fue desarrollado a principio de los 50 por Grace Hopper mientras desarrollaba el FLOWMATIC, lenguaje orientado al mundo empresarial.

Se considera al FORTRAN (*FORmula TRANslation*, 1954-57) como el primer lenguaje de alto nivel de uso generalizado. Está diseñado para el tratamiento de formulas matemáticas complejas. Con él aparecen los conceptos de variable, instrucciones condicionales y subrutinas independientes del programa principal.

En el año 59 aparece el COBOL (*COmmon Business-Oriented Language*) orientado a aplicaciones comerciales. Su diseño hace hincapié en las estructuras de datos de gran volumen, permitiendo un magnifico manejo para la organización de datos y manipulación de archivos. Además, presenta la característica de ser el lenguaje más similar al inglés ya que sus ordenes pueden escribirse como frases comunes (`TOTAL = IMPORTE + IVA`, en BASIC, `ADD IMPORTE TO IVA GIVING TOTAL`, en COBOL).

El lenguaje BASIC (*Beginners All-purpose Symbolic Instruction Code*) fue desarrollado a principio de los 60 y estaba dirigido a la enseñanza de la programación. Fue el primer lenguaje popularizado debido a la expansión de los microprocesadores en el ámbito doméstico que incorporaban el intérprete de BASIC en la ROM ya que es un lenguaje interpretado, lo que facilita la escritura y depuración de los programas por gente con pocos conocimientos metodológicos.

En la actualidad, los lenguajes más extendidos son el C (1972) y sus derivados, tanto en el entorno de desarrollo de sistemas (en él están desarrollados los sistemas operativos UNIX y LINUX) como en el de las aplicaciones, y los lenguajes orientados a objeto, con un cambio radical en la filosofía de la programación. Ya no se trata de dar una serie de ordenes consecutivas, sino que se manipulan objetos, que actuarán o no ante determinados eventos. Un programa orientado a objetos consta de un conjunto de objetos (ventanas, botones, zonas de entrada de texto, tablas, ficheros, etc.) interrelacionados a través de los eventos que soportan (implementados en el lenguaje) y los métodos desarrollados mediante programación para cada uno de ellos.

En la década de los 90 el C++ es el lenguaje orientado a objeto más popular, seguido por el JAVA (evolucionado del C++) que fue creado específicamente para programar en Internet, aunque puede ser usado para aplicaciones de propósito general.

Los lenguajes visuales (para programar en S.O. visuales como Windows) son el siguiente paso. Con ellos, y mediante programación orientada a objetos, se está programando en la actualidad (VisualBasic, Delphy, Objetive Cobol, C++ Builder, etc.)

8.3. LA TRADUCCIÓN.

Consiste en convertir un programa fuente (en texto ASCII) en ejecutable (en código máquina). Dependiendo del lenguaje, se utilizarán unas herramientas u otras para realizar el proceso. Estas herramientas (ensambladores, compiladores, linkadores o encuadernadores) forman parte del sistema operativo aunque vengan integradas en el paquete del lenguaje ya que el resultado que se obtiene con ellas es específico para un determinado procesador (ya que cada uno tiene su propio código máquina) y S.O.

En los lenguajes ensambladores, la traducción se realiza en un sólo paso ya que las ordenes del código fuente son equivalentes a las del código máquina, excepto que las primeras utilizan una notación mnemotécnica para facilitar el trabajo del programador.

En los lenguajes de alto nivel, la traducción puede hacerse con dos modalidades.

8.3.1 INTERPRETACIÓN.

Un interprete hace que el programa fuente vaya, sentencia a sentencia, traduciéndose y ejecutándose. El intérprete coge una sentencia, la analiza y traduce a código máquina y la ejecuta antes de proceder con la siguiente. La traducción no se almacena por lo que si la orden debe volver a ejecutarse, la traducirá de nuevo.

Entre los inconvenientes de este método cabe destacar:

- Si una orden ha de ejecutarse 100000 veces (bucles) la traducirá 100000 veces.
- Un simple error sintáctico sólo será detectado al ejecutarse la línea correspondiente. Si el error está en una bifurcación condicional, por ejemplo, en un bloque de instrucciones que sólo se ejecutarán en una determinada fecha, sólo lo detectaremos cuando se cumpla esa condición. Por ello, un programa nos puede dar un error a pesar de haberse ejecutado correctamente cientos de veces.
- Necesitan ejecutarse a través del interprete por lo que requieren el llamado entorno del lenguaje. Si trasladamos el programa a otra máquina tendremos que trasladar también el entorno.
- El número de órdenes y funciones que manipulan es bastante limitado

Las ventajas que presenta son:

- Los programas suelen ser cómodos de depurar ya que la ejecución se para al detectar cualquier error (sintáctico o de ejecución), permitiendo al programador depurarlo.
- Podemos parar la ejecución en cualquier punto, analizar y/o modificar los valores de los datos y reanudarla.
- Permiten ejecutar la mayoría de las órdenes aisladamente, desde la línea de comandos del entorno.
- El entorno suele ocupar poca memoria.

Todas estas características hacen que los lenguajes interpretados sean muy cómodos para el aprendizaje aunque engorrosos para la programación profesional.

Entre los lenguajes interpretados destacan el dBASE y el BASIC (que incluso iba incorporado en la ROM en los primeros ordenadores domésticos).

8.3.2 COMPILACIÓN.

Consiste en traducir la totalidad del fuente en una sola operación, generando como resultado un nuevo fichero que contendrá el programa ejecutable. Este último podrá ejecutarse en cualquier máquina con el mismo tipo de procesador, sin necesidad de que las herramientas utilizadas para la generación del programa estén presentes.

Aunque genéricamente se usa el término compilación para el proceso de convertir un fuente en ejecutable, el proceso consta de dos fases claramente diferenciadas:

COMPILACIÓN DEL FUENTE

Primero se hace la revisión léxica, sintáctica y semántica de cada una de las ordenes, es decir, comprobar que las ordenes corresponden a las que admite el lenguaje, que están estructuradas correctamente y que el orden y tipo de los parámetros sea el correcto.

A continuación, convierte las ordenes en texto ASCII a un código intermedio (similar al ensamblador en algunos casos o a código máquina en otros) generando con él el llamado programa objeto.

En cualquier caso, el objeto no es ejecutable directamente ya que le hace falta que se le incorporen las rutinas externas a las que hace llamadas, bien sean estas pertenecientes al usuario o al sistema. Las primeras suelen ser otros programas del usuario (que contienen rutinas o procesos requeridos por el programa) mientras las segundas forman parte de las llamadas librerías del lenguaje y contienen, entre otros elementos, las llamadas a las funciones básicas del sistema operativo o funciones del propio lenguaje.

Dependiendo de lenguaje, el objeto puede ser almacenado en forma de fichero en el disco o quedarse en memoria a la espera de que se inicie la segunda fase.

ENCUADERNACIÓN O LINKAJE.

En esta fase, se añaden al objeto los módulos objeto requeridos, ya sean de usuario o de las librerías del lenguaje. Así, de un conjunto de programas objeto se obtiene un único ejecutable.

Esta fase del proceso es la que determina en que tipos de procesador y/o sistema operativo será ejecutable el programa ya que es en este punto donde se incorporan las ordenes especificas del sistema para el manejo del hardware y del sistema de archivos.

Este proceso se estudió detalladamente en el apartado Montaje (página 25).

EL DEBUGGER

Es un módulo objeto que, linkado junto con el programa, permite hacer un seguimiento paso a paso de la ejecución del programa compilado. Las características y forma de funcionamiento del debugger dependen del lenguaje en si, pero todos presentan determinadas similitudes: parar un programa en un punto determinado, examinar los valores de los datos, análisis de la ejecución (trazo o *trace*).

En resumen, el debugger nos permite ejecutar un programa compilado de forma similar a como lo hace un programa en un lenguaje interpretado.

8.3.3 TRADUCCIÓN MIXTA.

Algunos lenguajes utilizan indistintamente la interpretación y la compilación. De esta forma, se obtienen las ventajas de la interpretación en la fase de desarrollo del programa y las de la compilación en la fase de ejecución. Este es el caso del BASIC, que puede ser compilado, el dBASE, compilable con Clipper, o de algunos lenguajes que presentan un entorno para el desarrollo (Visual Basic) que permiten interpretar el programa o compilarlo, a voluntad del programador.

9. CARGA Y MONTAJE.

Una aplicación está formada por una serie de módulos compilados o ensamblados en forma de código objeto que se montan juntos para resolver las referencias entre módulos. Al mismo tiempo, se resuelven las referencias a rutinas de biblioteca. Las rutinas de biblioteca pueden estar incorporadas en el programa o ser referenciadas como código compartido que debe suministrar el sistema operativo en el momento de la ejecución.

Para la creación de un proceso activo, tras el montaje, debemos cargar un programa en memoria principal y crear una imagen del proceso.

9.1. MONTAJE

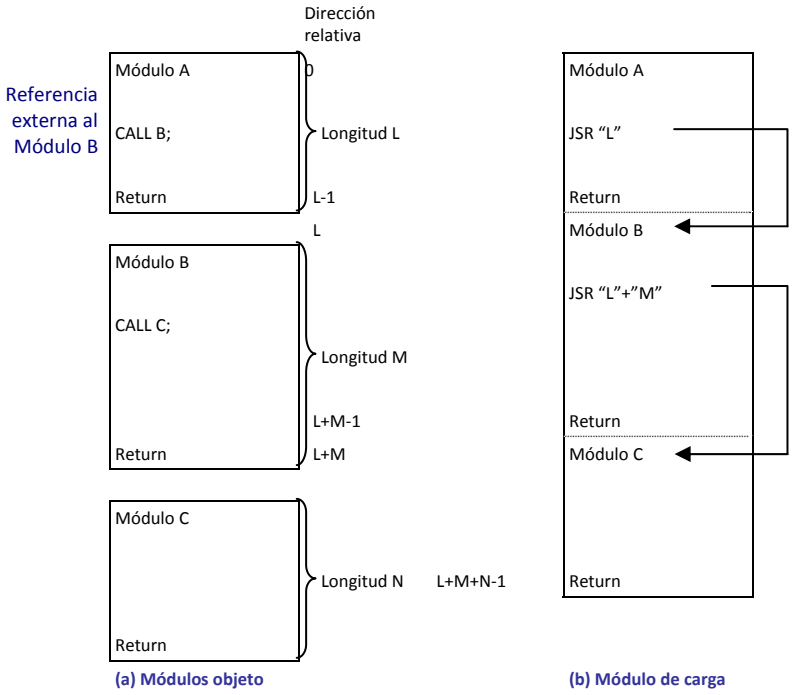
La función de un montador consiste en tomar como entrada una colección de módulos objeto y generar un módulo de carga que conste de un conjunto integrado de módulos de programa y de datos para el cargador. En cada módulo objeto pueden haber referencias a direcciones situadas en otros módulos. Cada una de estas referencias puede expresarse sólo simbólicamente en un módulo objeto n montado. El montador crea un único módulo de carga que es la concatenación de todos los módulos objeto. Cada referencia interna de un módulo debe cambiarse de dirección simbólica a una referencia a una posición dentro del módulo de carga total. Por ejemplo, si un módulo A contiene una llamada a un procedimiento de un módulo B, cuando se combinan estos módulos en el módulo de carga, esta referencia simbólica al módulo B se cambia por una referencia específica a la posición del punto de entrada de B en el módulo de carga.

9.1.1 EDITOR DE MONTAJE.

La esencia del montaje de direcciones dependerá del tipo de módulo de carga a crear y de cuándo se produzca el montaje (ver tabla a continuación). Si, como suele ser el caso, se desea un módulo de carga reubicable, el montaje se realiza generalmente de la siguiente forma. Cada módulo objeto compilado o ensamblado se crea con referencias relativas al comienzo del módulo. Todos estos módulos se unen en un único módulo de carga reubicable, junto con todas las referencias relativas al origen del módulo de carga. Este módulo puede usarse como entrada para una carga reubicable o para una carga dinámica durante la ejecución.

Los montadores que generan módulos de carga reubicables se conocen a menudo como editores de montaje.

Tiempo de montaje	Función
Tiempo de programación	No se permiten referencias a programas o datos externos. El programador debe situar dentro del programa el código fuente de todos los subprogramas que sean referenciados
Tiempo de compilación o ensamblaje	El ensamblador debe ir a buscar el código fuente de cada subrutina que sea referenciada y ensamblarlas como una unidad.
Creación del módulo de carga	Todos los módulos objeto se han ensamblado usando direcciones relativas. Estos módulos se montan juntos y todas las referencias se declaran de nuevo relativas al origen del módulo de carga final.
Montaje dinámico en tiempo de carga	Las referencias externas no se resuelven hasta que el módulo de carga sea ubicado en memoria principal. En ese momento, los módulos montados y referenciados dinámicamente se añaden al módulo de carga, enviando el paquete completo a la memoria principal o virtual.
Montaje dinámico en tiempo de ejecución	Las referencias externas no se resuelven hasta que el procesador ejecute las llamadas externas. En ese momento, se interrumpe el proceso y se monta el programa llamador con el módulo que se necesita.



Funcionamiento del editor de montaje

9.1.2 MONTADOR DINÁMICO.

Así como en la carga, es posible aplazar algunas funciones de montaje. El término montaje dinámico se emplea para referirse a la práctica de retrasar el montaje de algunos módulos externos hasta después de que el módulo de carga se haya creado. Así pues, el módulo de carga contiene referencias no resueltas a otros programas. Estas referencias pueden resolverse tanto en la carga como en la ejecución.

En el montaje dinámico en tiempo de carga se suceden las siguientes etapas. El módulo de carga (módulo de aplicación) se trae a memoria principal. Cualquier referencia a un módulo externo (módulo destino) hace que el cargador busque el módulo destino, lo cargue y modifique las referencias a direcciones relativas de memoria desde el comienzo del módulo de aplicación. Existen varias ventajas en este enfoque sobre el que podría llamarse carga estática, como son las siguientes:

- Resulta más fácil incorporar cambios o actualizar versiones del módulo destino, lo que puede constituir una utilidad del sistema operativo o alguna otra rutina de propósito general. En el montaje estático, cualquier cambio en el módulo soporte requeriría volver a montar el módulo de aplicación por completo. Esto no sólo es ineficiente, sino que puede ser imposible en determinadas circunstancias. Por ejemplo, en el campo de los ordenadores personales, la mayoría del software comercial se entrega en forma de módulo de carga; no se entregan las versiones fuente y objeto.
- Tener el código destino en un fichero de montaje dinámico prepara el terreno para la compartición automática de código. El sistema operativo puede darse cuenta que más de una aplicación está empleando el mismo código destino, puesto que habrá cargado y montado dicho código. Esta información puede usarse para cargar una única copia del código destino y montarla en ambas aplicaciones, en vez de tener que cargar una copia para cada aplicación.
- Resulta más fácil para los productores independientes de software ampliar la funcionalidad de un sistema operativo muy empleado. Un productor de software puede proponer una nueva función que sea útil para varias aplicaciones y empaquetarla como un módulo de montaje dinámico.

En el montaje dinámico en tiempo de ejecución, parte del montaje se pospone hasta el momento de la ejecución. Las referencias externas a los módulos destino permanecen en el programa cargado. Cuando se realiza una llamada a un módulo ausente, el sistema operativo localiza el módulo, lo carga y lo monta en el módulo llamador.

9.2. CARGA

El módulo de carga (que contiene todos los módulos montados juntos) se sitúa en la memoria principal, comenzando en la posición x. El sistema operativo deberá poder conseguir las direcciones de la información de control del proceso y de la pila de ejecución, así como el punto de partida para comenzar la ejecución del programa. Además el procesador debe ocuparse de las referencias a memoria dentro del programa. Las instrucciones de bifurcación deben contener la dirección que haga referencia a la instrucción que se vaya a ejecutar a continuación. Las instrucciones que hagan referencia a datos deben contener la dirección del byte o de la palabra de datos referenciada. De algún modo, el hardware del procesador y el software del sistema operativo deben ser capaces de traducir las referencias a memoria encontradas en el código del programa a las direcciones físicas reales que reflejen la posición actual del programa en memoria principal. En general se pueden aplicar tres métodos:

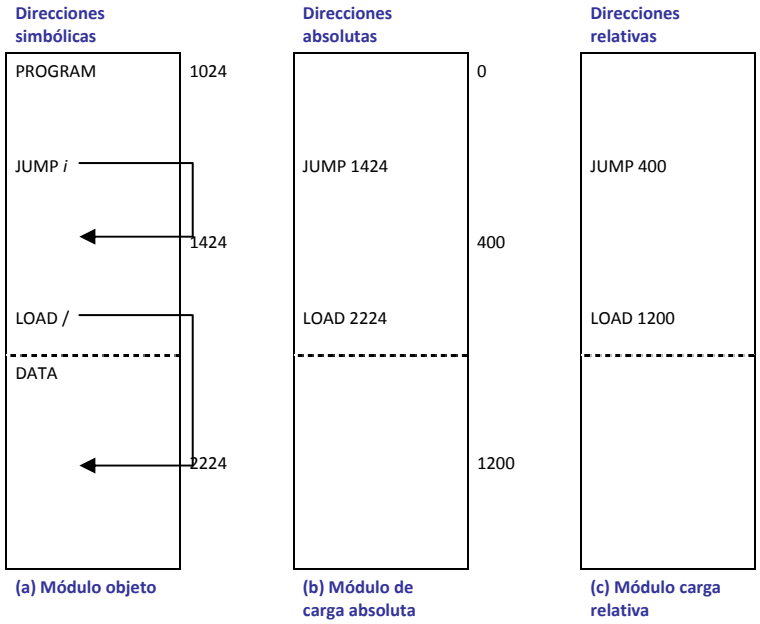
9.2.1 CARGA ABSOLUTA.

La carga absoluta necesita que el módulo de carga ocupe siempre la misma posición de memoria principal. Así pues, todas las referencias del módulo de carga para el cargador deben ser direcciones específicas o absolutas en memoria principal. Por ejemplo, si X (posición de comienzo) es la posición 1024, la primera palabra del módulo de carga destinado a esa región de memoria tendrá la dirección 1024. La asignación de direcciones específicas a las referencias a memoria de un programa puede ser realizada tanto por el programador como en tiempo de compilación o ensamblaje (ver tabla siguiente). Con el primer método se tienen varias desventajas. En primer lugar, todos los programadores tendrán que conocer la estrategia de asignación deseada para situar los módulos en memoria principal. En segundo lugar, si se hace alguna modificación en el programa que suponga inserciones o borrados en el cuerpo del módulo, tendrán que cambiarse todas las direcciones. Por consiguiente, es preferible permitir que las referencias a memoria dentro de los programas se expresen simbólicamente y que se resuelvan en el momento de la compilación o el ensamblaje. Todas las referencias a una instrucción o elemento de datos se representan inicialmente por un símbolo. Cuando se prepara el módulo para la entrada a un cargador absoluto, el ensamblador o el compilador convertirán todas estas referencias a direcciones específicas (en el ejemplo siguiente, para cargar el módulo en la posición de comienzo 1024).

Tiempo de enlace	Función
Tiempo de programación	El programador especifica directamente todas las direcciones físicas reales en el propio programa.
Tiempo de compilación o ensamblaje	El programa contiene referencias simbólicas a direcciones y el compilador o el ensamblador las convierten en direcciones físicas reales.
Tiempo de carga	El compilador o ensamblador genera direcciones relativas. El cargador traduce éstas a direcciones absolutas en el instante de la carga del programa.
Tiempo de ejecución	El programa cargado conserva direcciones relativas. El hardware del procesador las convierte dinámicamente en direcciones absolutas.

9.2.2 CARGA REUBICABLE.

La desventaja de asociar las referencias a memoria a direcciones específicas previas a la carga es que el módulo de carga resultante sólo puede situarse en una región de memoria principal. Sin embargo, cuando varios programas comparten la memoria principal, puede no ser conveniente decidir por adelantado en qué región de memoria debe cargarse un módulo en particular. Es mejor tomar esa decisión en el momento de la carga. Así pues, se necesita un módulo de carga que pueda ubicarse en cualquier posición de la memoria principal.



Para satisfacer este nuevo requisito, el ensamblador o el compilador no generarán direcciones reales de memoria principal (direcciones absolutas) sino direcciones relativas a algún punto conocido, tal como el comienzo del programa. Esta técnica se ilustra en la parte c) de la figura siguiente. Al comienzo del módulo de carga se le asigna la dirección relativa 0 y todas las demás referencias del módulo se expresan en relación al comienzo del módulo.

Con todas las referencias a memoria expresadas de forma relativa, situar los módulos en la posición deseada se convierte en una tarea sencilla para el cargador. Si el módulo va a ser cargado comenzando por la posición x, el cargador simplemente sumará x a cada referencia a memoria a medida que cargue el módulo en memoria. Para ayudar en esta tarea, el módulo de carga debe incluir información que indique al cargador dónde están las referencias a direcciones y cómo se interpretan (generalmente, de forma relativa al comienzo del programa pero también es posible que sean relativas a algún otro punto del programa, como la posición actual). El compilador o el ensamblador preparan este conjunto de información que se conoce normalmente como diccionario de reubicación.

9.2.3 CARGA DINÁMICA EN TIEMPO DE EJECUCIÓN.

Los cargadores con reubicación son habituales y ofrecen ventajas obvias en relación con los cargadores absolutos. Sin embargo, en un entorno multiprogramado, incluso sin memoria virtual, el esquema de carga reubicable resulta inadecuado. Se ha hecho referencia a la necesidad de cargar y descargar las imágenes de procesos de memoria principal para maximizar la utilización del procesador. Para maximizar el uso de la memoria principal, sería conveniente volver a cargar una imagen de un proceso en posiciones diferentes para diferentes instantes de tiempo. Así pues, un programa cargado puede ser expulsado a disco y vuelto a cargar en una posición distinta. Este procedimiento resultaría imposible si las referencias a memoria hubieran estado asociadas a direcciones absolutas en el momento inicial de carga.

Una alternativa consiste en aplazar el cálculo de direcciones absolutas hasta que realmente se necesitan durante la ejecución. Con este propósito, el módulo de carga se trae a memoria con todas las referencias de forma relativa. La dirección absoluta no se calcula hasta que se ejecuta una instrucción. Para asegurar que esta función no degrada el rendimiento, debe realizarse por medio de un hardware especial del procesador, en vez de por software.

El cálculo de direcciones dinámico proporciona una completa flexibilidad. Un programa puede cargarse en cualquier región de memoria principal. Más tarde, la ejecución del programa puede interrumpirse y el programa ser descargado de memoria principal para posteriormente ser cargado en una posición diferente.

10. SEGURIDAD.

El nombre genérico del conjunto de herramientas diseñadas para proteger los datos y frustrar a los piratas informáticos (*hackers*) es el de seguridad de ordenadores.

10.1. AMENAZAS A LA SEGURIDAD.

10.1.1 TIPOS DE AMENAZAS.

Los tipos de amenazas a la seguridad de un sistema de ordenadores o una red se caracterizan mejor contemplando la función del sistema como suministrador de información. En general, se produce un flujo de información desde un origen, como un archivo o una región de memoria principal, hacia un destino, como otro archivo o un usuario. Las categorías generales de las amenazas son:

- **Interrupción**: Se destruye un elemento del sistema o se hace inasequible o inútil. Ésta es una amenaza a la disponibilidad (los elementos de un sistema de ordenadores deben estar disponibles a grupos autorizados). Como ejemplos se incluyen la destrucción de una pieza de hardware, como un disco duro, el corte de una línea de comunicaciones o la inutilización del sistema de gestión de ficheros.
- **Intercepción**: Una parte no autorizada consigue acceder a un elemento. Esto es una amenaza a la confidencialidad. La parte no autorizada puede ser una persona, un programa o un ordenador. Como ejemplos se incluyen la interceptación de las conexiones telefónicas para capturar datos de una red y la copia ilícita de archivos o programas.
- **Modificación**: Una parte no autorizada no sólo consigue acceder, sino que falsifica un elemento. Ésta es una amenaza a la integridad. Se pueden citar como ejemplos el cambio de valores en un archivo de datos, la alteración de un programa para que se comporte de manera diferente o la modificación del contenido de los mensajes transmitidos en una red.
- **Invención**: Una parte no autorizada inserta objetos falsos en el sistema. Ésta es también una amenaza de integridad.

10.1.2 AMENAZAS A CADA UNO DE LOS ELEMENTOS DEL SISTEMA.

HARDWARE.

La amenaza principal al hardware se produce en el campo de la disponibilidad. Comprende daños accidentales y deliberados a los equipos, así como el hurto. Hacen falta medida de seguridad físicas para hacer frente a estas amenazas.

SOFTWARE.

Su amenaza principal es la disponibilidad. El software, en especial el de aplicación, es asombrosamente fácil de eliminar. Una gestión cuidadosa de la configuración del software, que incluye la realización de copias de reserva de las versiones más recientes, puede conservar una alta disponibilidad.

Un problema más difícil de afrontar es la modificación del software que provoca que un programa siga funcionando pero se comporte de forma diferente que antes. Los virus informáticos entran dentro de esta categoría y se tratarán posteriormente en este capítulo.

Un problema final es la confidencialidad del software. Aunque se facilitan ciertas contramedidas, por lo general el problema de la copia no autorizada de software no se ha resuelto.

DATOS.

El interés de la seguridad con respecto a los datos es amplio, abarcando la disponibilidad, la confidencialidad y la integridad. En el caso de la disponibilidad, la preocupación es la destrucción de los archivos de datos, lo que puede ocurrir accidentalmente o como consecuencia de una mala intención. Para asegurar la disponibilidad de los datos hay que procurar copiarlos para poder recuperarlos en caso de problema (puede hacerse en disquetes, discos removibles, cintas magnéticas, etc.). Existen varias estrategias de salvaguarda, que ofrecen distintos niveles de seguridad:

- Salvaguarda simple en un único soporte (juego de disquetes, por ejemplo): ofrece una seguridad limitada. En efecto, antes de hacer la salvaguarda hay que borrar el contenido de los disquetes. Si se produjera un problema durante la salvaguarda, se tienen bastantes probabilidades de perder todos los datos.
- La salvaguarda múltiple (en dos juegos de disquetes, por ejemplo) ofrece más seguridad. Si se produjera un problema durante la salvaguarda, el segundo juego estaría intacto (con la condición de que no se empiece por borrar los dos juegos antes de hacer la salvaguarda). Sin embargo, este sistema no protege ante errores lógicos que no suprimen los datos pero los contaminan. Así, si se hubiera borrado por error el contenido de un fichero sin borrar el propio fichero, en la salvaguarda se copiaría un fichero vacío. Cuando quisiéramos darnos cuenta, sería demasiado tarde.
- Para evitar el problema anterior, puede hacerse una salvaguarda los días pares y otra los días impares. El problema del borrado del contenido de un fichero no tendría entonces consecuencias si lo detectamos antes de 24 horas.
- Para aumentar la seguridad, pueden conservarse las salvaguardas durante más tiempo. Por ejemplo, puede hacerse una salvaguarda diferente para cada día de la semana. Se podrán así recuperar los datos en un estado anterior si fuera necesario. Todavía puede aumentarse la seguridad si se conservara la salvaguarda de un día (el viernes, por ejemplo) durante cierto tiempo, con el fin de incrementar las posibilidades de descubrir los problemas antes de que sea demasiado tarde.

La preocupación obvia de la confidencialidad es, si duda, la lectura no autorizada de archivos o bases de datos.

Finalmente, la integridad de los datos es una preocupación fundamental de la mayor parte de instalaciones. Las modificaciones de archivos de datos pueden tener consecuencias poco trascendentes o desastrosas.

REDES Y LÍNEAS DE COMUNICACIONES.

Los sistemas de comunicaciones se utilizan para transmitir datos. Por tanto, las importantes medidas de seguridad aplicadas a los datos (disponibilidad, seguridad e integridad) también se aplican a la seguridad de las redes.

En este contexto las amenazas se clasifican en pasivas (escuchas a escondidas o supervisión de las transmisiones de una organización con la intención de obtener la información que se está transmitiendo) y activas (alteración del flujo de datos o creación de un flujo falso por ejemplo alterando el contenido de los mensajes, inhibiendo el servicio de mensajería o fingiendo ser una entidad diferente suplantando a un usuario autorizado). Las amenazas pasivas son muy difíciles de detectar porque no acarrean alteración alguna de los datos. Hay que hacer frente a estas amenazas mediante la prevención. El objetivo sin embargo con respecto a los ataques activos es detectarlos y recuperarse de cualquier interrupción o retardo causado.

10.2. PRINCIPIOS DE DISEÑO DE LAS MEDIDAS DE SEGURIDAD.

- **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 sólo por permiso explícito; por omisión deberían ser "sin acceso".
- **Ahorro de mecanismos**: Los mecanismos de seguridad deben ser tan pequeños y simples como sea posible y estar incluidos en los niveles más bajos del sistema.
- **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 autorizan el acceso. Si los mecanismos no son fáciles de usar, probablemente no van a ser usados o lo serán de forma incorrecta.
- **Mediación total:** Cada acceso debe ser cotejado con la información de control de acceso. Hay que comprobar la autorización en cada momento.
- **Diseño abierto**: La seguridad del sistema no debe depender de guardar en secreto el diseño de sus mecanismos. La suposición de que los intrusos no conocen el funcionamiento del sistema sólo sirve para que los diseñadores se engañen a sí mismos.

10.2.1 PROTECCIÓN.

PROTECCIÓN DE MEMORIA.

En un entorno de multiprogramación, la protección de la memoria principal es fundamental. El interés no es sólo la seguridad, sino también el funcionamiento correcto de los diversos procesos que estén activos. Si un proceso puede escribir inadvertidamente en el espacio de memoria de otro proceso, este último puede no ejecutarse correctamente.

La separación del espacio de memoria de los diversos procesos se lleva a cabo fácilmente con un esquema de memoria virtual. La segmentación, paginación o la combinación de ambas proporciona un medio eficaz de gestión de la memoria principal.

Si se persigue un aislamiento total, el sistema operativo simplemente debe asegurar que cada segmento o cada página es accesible sólo para el proceso al que está asignada. Esto se lleva a cabo fácilmente exigiendo que no haya entradas duplicadas en las tablas de páginas o segmentos. Si se va a permitir la compartición, el mismo segmento o página puede ser referenciado en más de una tabla. Este tipo de compartición se consigue mejor en un sistema que soporta segmentación o una combinación de segmentación y paginación. En tal caso, la estructura del segmento es visible a la aplicación y la aplicación puede declarar segmentos individuales como compartibles o no compartibles.

CONTROL DE ACCESO ORIENTADO AL USUARIO.

La técnica más habitual de control de acceso al usuario, en un sistema de tiempo compartido o en un servidor, es en la conexión del usuario, que requiere un identificador de usuario (ID) y una contraseña. Este sistema no es un método muy fiable ya que los usuarios pueden olvidar sus contraseñas y pueden revelarlas accidental o deliberadamente. Los piratas informáticos son muy habilidosos en adivinar los ID de usuarios específicos, como el personal de control o de gestión del sistema.

El problema del control de acceso a los usuarios se complica en las redes de comunicaciones. El diálogo de conexión debe tener lugar a través del medio de comunicación y las escuchas se convierten en una amenaza potencial.

El control de acceso al usuario en sistemas distribuidos puede ser centralizado o descentralizado. Con un enfoque centralizado, la red proporciona un servicio de conexión para determinar a quién se le permite usar la red y a qué se le permite conectarse.

El control de acceso descentralizado considera la red como un enlace transparente de comunicaciones y el procedimiento usual de conexión lo lleva a cabo el servidor de destino. En muchas re-

des, pueden emplearse dos niveles de control de acceso. Los servidores individuales pueden estar provistos de un servicio de conexión que proteja las aplicaciones y recursos específicos del servidor.

Además, la red en conjunto puede ofrecer una protección para restringir el acceso a la red a los usuarios no autorizados.

CONTROL DE ACCESO ORIENTADO A LOS DATOS.

Asociado a cada usuario, puede haber un perfil de usuario que especifique las operaciones y los accesos a archivos permisibles. El sistema operativo puede hacer valer unas reglas en función del perfil del usuario. El sistema gestor de la base de datos, sin embargo, debe controlar el acceso a registros específicos o incluso partes de un registro. Por ejemplo, puede permitirse que cualquier administrador obtenga un listado del personal de una compañía, pero solamente unos individuos elegidos pueden tener acceso a la información de salarios.

Un modelo general de control de acceso, ejercido por un sistema gestor de archivos o bases de datos, es el de una matriz de acceso. Los elementos básicos del modelo son los siguientes:

- **Sujeto**: Una entidad capaz de acceder a los objetos (cualquier usuario o aplicación).
- **Objeto**: Cualquier elemento cuyo acceso debe controlarse (archivos, partes de archivos, programas y segmentos de memoria).
- **Derecho de acceso**: La manera en que un sujeto accede a un objeto (leer, escribir, ejecutar).

Una dimensión de la matriz de acceso consta de los sujetos identificados que pueden intentar acceder a los datos (usuarios, grupos de usuarios, aplicaciones, terminales...) y la otra dimensión enumera los objetos accesibles (campos, registros, archivos...). Cada entrada de la matriz indica los derechos de acceso del sujeto al objeto.

En la práctica las matrices de acceso se implementan por descomposiciones en columnas o filas. La descomposición en columnas da lugar a listas de control de acceso. Así pues, para cada objeto, una lista de control de acceso (ACL, *Access Control List*) expresa los usuarios y sus derechos de acceso permitidos. La descomposición por filas da lugar a etiquetas de capacidades (*capabilities*). Una etiqueta de capacidades especifica los objetos y las operaciones autorizadas para un usuario.

10.2.2 INTRUSOS.

Una de las dos amenazas más conocidas a la seguridad (la otra es la de los virus) es el intruso, conocido en general como pirata (*hacker* o *cracker*). El objetivo de los intrusos es obtener acceso a un sistema o aumentar el conjunto de privilegios accesibles en un sistema. En la mayoría de los casos pretenden obtener información en forma de una contraseña de usuario con la que conectarse al sistema y hacer ejercicio de los privilegios convenidos con el usuario legítimo.

Normalmente, un sistema debe mantener un archivo que asocia una contraseña a cada usuario autorizado. Este archivo debe protegerse de dos maneras: cifrando las contraseñas y limitando el acceso al archivo de contraseñas a una o muy pocas cuentas.

10.2.3 VIRUS.

En 1983, Fred Cohen (estudiante de la Universidad de California) empleó 8 horas en confeccionar un experimento para un seminario sobre seguridad informática. El diseño consistía en un programa que era capaz de modificar a otros, incluyéndose en ellos, y replicarse en el resto de los programas del ordenador. En un tiempo medio de media hora, el "experimento" se adueño de todos los equipos en que fue implantado. Poco después Len Adleman denominó a este tipo de programas **VIRUS**. Existen tres tipos básicos de virus:

- **Gusanos**: es un programa diseñado para reproducirse a través de las redes informáticas. Su misión principal es colapsar el sistema infectado por sobrecarga de sus recursos.

- **Caballos de Troya**: es un programa que modifica o destruye la información mientras simula realizar otra tarea. Suelen presentarse en forma de juegos y no se reproducen.
- **Virus**: combinan el poder reproductivo de los gusanos con el destructivo de los caballos de Troya. Su misión principal es reproducirse lo máximo posible antes de ser detectados o iniciar su fase destructiva.

Ya que la mayoría de los virus existentes son del tercer tipo, vamos a hablar de estos. Hay virus que tienen por misión eliminar otros virus, otros que simplemente muestran un mensaje o acción inofensiva en pantalla y otros que después de hacer su "gracia" se autodestruyen, pero la gran mayoría son más dañinos para el ordenador que el virus de la gripe para el ser humano.

CICLO DE LOS VIRUS.

Todos los virus nacen por obra y gracia de un programador, aunque hay virus que son mutaciones desarrolladas por otros programadores o mutaciones autoprovocadas por el propio virus con el fin de ocultarse. Están diseñados para propagarse y destruir la información de un determinado tipo de hardware ya que, para conseguir sus propósitos, deben manipular éste. Un 90% de los virus está diseñado para PCs.

Se transmiten de un ordenador a otro a través de la copia o uso de ficheros contaminados. Estos ficheros pueden estar en un disquete o ser adquiridos directamente de otros equipos mediante conexiones por red o Internet.

Dentro del mismo ordenador, al ejecutar un programa infectado, el virus se carga en la RAM y pasa a contaminar metódicamente todos los programas que ejecutemos a continuación. Al apagar el ordenador, el virus se pierde, pero volverá a cargarse en memoria en cuanto ejecutemos uno de los programas infectados.

Un virus está en latencia cuando está presente en un ordenador pero no en memoria ya que no puede replicarse. También llamamos latencia al periodo en que los virus se dedican exclusivamente a reproducirse.

Se activan bien sea en una fecha dada, como el Barrotes el 5 de enero, al reproducirse un determinado número de veces o al encender el ordenador un número dado de veces, como el Anti-Telefónica, que se activa al encender el ordenador 300 veces.

TIPOS DE VIRUS.

Virus de fichero.

Utilizan los ficheros ejecutables (`.EXE`, `.COM`, `.SYS`, `.OVL`, ...) como medio de transmitirse y tomar el control. Si se instalan en un programa de poco uso, no se carga en RAM hasta que lo ejecutemos. Si es un virus de acción directa (tipo caballo de Troya), ya está hecho el daño.

Por el contrario, si es del tipo de virus residente, lo primero que hace es ver si debe activarse (si es 5 de enero en el caso del Barrotes). Si no es el momento adecuado, se queda en RAM dedicándose a añadirse a todos los programas que ejecutemos, entre ellos el intérprete de comandos (`COMMAND.COM`) que se carga en RAM nada mas encender el ordenador. A partir de aquí, la infección es exponencial.

Un síntoma de infección viral es el aumento del tamaño de los ficheros. Algunos virus de fichero se instalan en el fichero infectado sobreescribiendose, por lo que el programa queda con el mismo tamaño pero inutilizable. Son los llamados virus de sobreescritura.

Los virus de compañía crean un nuevo fichero con el mismo nombre que el que hemos ejecutado pero con la extensión `.COM` (los `.COM` tienen prioridad de ejecución sobre los `.EXE`) en el que instala una copia suya. A continuación le pone el atributo de oculto para que no se vea.

Los virus compresores se dedican a comprimir el fichero infectado. Estos pierden tamaño y, además, es más difícil limpiar el virus ya que éste también queda comprimido.

Un tipo raro de virus son los virus de enlace o de directorio que modifican el directorio cambiando el número del primer cluster del fichero ejecutado por el de la ubicación del virus y situando en un pseudodirectorio la dirección real. No se crean nuevas copias del virus, sino que hace que todos los ficheros infectados pasen primero por la única copia del virus y posteriormente, a través del pseudodirectorio, ejecuta el fichero.

Virus de BOOT.

Se instalan en el BOOT y la tabla de particiones de los discos. Se contagia al arrancar con un disquete contaminado (aunque no sea de arranque). Una vez instalado en el H.D., se carga en RAM nada mas encender el ordenador y se dedica a contagiar todos los disquetes que usemos. Son altamente dañinos pues al activarse suelen destruir la FAT.

Virus de macro.

Son los más abundantes hoy en día. No infectan programas sino ficheros de datos de OFFICE (suite de Microsoft) que, al contener macros, es como si contuviesen ficheros ejecutables. Dado que con el Word de Office se pueden crear paginas WEB de Internet y se suelen usar en el correo electrónico (además, el gestor de correo más popular, Outlook, también es de esta suite), su difusión se ha ampliado mucho en poco tiempo.

CAMUFLAJE DE LOS VIRUS.

Con el fin de dificultar su localización a los antivirus, los virus emplean algunos trucos como el ocultamiento (*stealth*) que consiste en suministrar al antivirus una copia del fichero que quiere examinar que no esté infectada (sólo si el virus está en RAM). Con el *tunneling* o sobrepasamiento intentan evitar que un antivirus residente les localice. El autoencriptamiento hace que cada copia del virus tenga un aspecto diferente con el fin de evitar su localización (pero tienen una rutina de encriptación común). El polimorfismo es un autoencriptamiento que cambia periódicamente la rutina de encriptación. El *armouring* incluye en el virus rutinas para evitar que pueda ser examinado de cara a la obtención de un antivirus.

10.2.4 LOS ANTIVIRUS.

Son programas que examinan la RAM, el sector de arranque y los ficheros susceptibles de ser infectados en busca de virus. Para ello utilizan métodos como búsqueda de determinadas cadenas de datos (el Viernes 13-B tiene la cadena `WARNING! THIS PROGRAM IS INFECTED WITH VIRUS-B`), búsquedas deductivas (examen del tamaño, fecha y hora de los ficheros) y la búsqueda heurística (anomalías o “cosas raras” en los ficheros) que, además, nos permite localizar virus nuevos (a cambio de un montón de falsas alarmas).

Salvo con la búsqueda heurística, sólo son capaces de detectar y eliminar (no siempre) los virus ya conocidos. Teniendo en cuenta que cada mes aparecen cientos de virus nuevos, es conveniente tener un antivirus lo mas actualizado posible (uno de hace 3 meses se considera obsoleto).

LAS VACUNAS.

Es un método por el que se toman datos de los ficheros que tenemos con el fin de poder determinar en un futuro si un fichero está infectado. El antivirus examina los datos actuales del fichero con los que contiene la vacuna. Si hay anomalías, procede a realizar un examen a fondo en búsqueda de virus.

ANTIVIRUS RESIDENTES O CENTINELAS.

Se cargan en memoria nada mas encender el ordenador. Examinan cualquier acceso a un disquete o a un fichero no vacunado. Si detectan un virus, bloquean el sistema o pasan a la ejecución del antivirus.

Presentan como inconvenientes la ocupación de RAM y el ralentizamiento del sistema. Además, previene contra los virus conocidos pero no contra los nuevos. Aun así, es conveniente tener uno instalado, sobre todo en los equipos que son manejados por varios usuarios o los conectados a redes o Internet.

LOS VIRUS E INTERNET.

El "contagio" de un virus de un ordenador a otro se puede dar por múltiples motivos: porque alguien le ha pasado un disquete con un fichero contaminado, porque alguien le dejó un CD-ROM contaminado, o bien, porque accedió a algún lugar de Internet donde existía un fichero o recibió un e-mail contaminado. Todo es lo mismo. El mecanismo es exactamente igual: un fichero contaminado llega a nuestra máquina. Si no ejecutamos el programa no pasará nada, si lo ejecutamos o abrimos, nos contaminaremos.

En Internet existen muchos mecanismos para poder hacernos con un fichero contaminado, ya que existen múltiples formas de transmitirlo. El más inmediato es que alguien nos lo envíe junto con un mensaje de correo electrónico. El mensaje no es peligroso porque es sólo texto y no se puede ejecutar. Hay que tener cuidado sin embargo con los ficheros que puedan venir adjuntados al mensaje. Sólo puede haber problema con los ficheros que tengan capacidad de ejecución, por lo que, en teoría, podemos abrir cualquier fichero que no tenga partes ejecutables y, por lo tanto, ver una foto si ésta viene en cualquiera de sus formatos estándar (BMP, JPEG, GIF, etc.), un vídeo (.AVI, .MOV, .MPG, etc.), un fichero de texto (.TXT). También hayt que tener cuidado con los documento de Word que, pese a ser un fichero que en principio sólo tiene texto, esto no es del todo correcto, pues Word permite crear macros y, las macros no son más que programas y, como consecuencia de esto, vulnerables ante los virus. Este es el caso del famoso *Melissa* que era una macro dentro de un documento Word.

Como medida de protección es aconsejable no abrir ningún fichero procedente de ningún lugar que pueda poseer una parte activa, como los documentos Word (.DOC), las hojas de cálculo Excel (.XLS), los dibujos de CorelDRAW (.DRW) y, por supuesto los ejecutables directamente (.EXE, .COM).

También es fundamental no abrir ningún correo cuyo remitente nos sea desconocido.

Es cierto que, dada la diversidad de formas de transferir ficheros en Internet, los métodos de contaminación aumentan proporcionalmente. Hemos visto el peligro que puede existir con el correo electrónico pero, como no es la única vía para transferir un fichero desde Internet, tampoco es la única forma de posible contagio.

Una forma usual de quedar contaminado con un virus es seguir un enlace en el navegador que nos lleva a la descarga de un fichero. Si éste está contaminado, podremos ser infectados. Evidentemente ese método no difiere en nada con respecto a la copia de un fichero desde un disquete, un CD-ROM o una unidad de red.

Una forma de contagio sofisticada y nueva consiste en hacer uso de los *scripts* que se añaden a las páginas web, que no es más que un fichero HTML que originariamente sólo contenían texto pero que en las últimas versiones se puede incluir texto ejecutable, es decir, *scripts* en Visual Basic Script, JavaScript o Jscript. Al llevar una parte ejecutable, volvemos a correr el riesgo de poder ser contaminados. El funcionamiento de este tipo de virus es algo complejo, aunque se basan en el mismo esquema que el resto de los virus. Básicamente existe una función que se autoejecuta cuando el navegador abre la página. A partir de ese momento se puede llegar a obtener un virus.

Por supuesto, no hay que descartar mecanismos como ICQ, IRC, FCT, etc., ya que todos ellos pueden efectuar cargas de ficheros y, además, en algunos casos los propios programas pueden almacenar *scripts* de automatización, aumentando así las posibilidades de obtener virus.

Los fanáticos del mundo de los virus no paran en su carrera por obtener cada día virus más difícilmente localizables y con mayor poder destructor. Incluso se dice que las guerras de este siglo que viene podrían consistir en bombardeos electrónicos. Se prevé que en unos años absolutamente todos los dispositivos estén conectados a la Red, desde el ordenador del trabajo hasta el reloj de la muñeca, pasando por sistema de control de la vivienda, el coche y cualquier otra cosa que podamos imaginar. Así que esta posibilidad, que ahora parece absurda, quizá en un futuro pueda hacer más daño de lo que parece.

En este sentido se está trabajando y ya se habla de bombas "personalizadas". Éstas son virus pero que, en lugar de activarse en cualquier ordenador bajo algunas condiciones, sólo se activan en el equipo para el que han sido diseñadas. El mecanismo se basa en el principio que dice que cada PC de la red tiene que tener un número exclusivo (el IP). Si se programa de algún modo un virus que se expanda por la Red buscando el ordenador de "destino" y únicamente estalle en el que tiene el número para el que fue programado, tendríamos un virus personalizado, algo parecido a los misiles pero, en este caso, con formato electrónico.

DIRECCIONES ÚTILES.

http://www.seguridadenlared.org/

http://www.osi.es/es/conoce-los-riesgos

http://www.inteco.es/home/instituto_nacional_tecnologias_comunicacion/

Investiga 5:

1. Define malware y, al menos, tres tipos de malware.
2. Define crimeware y, al menos, tres tipos de crimeware.
3. Define fraude-on-line y, al menos, cuatro modalidades de fraude-on-line.