

1-. INTRODUCCIÓN.

Un computador sin software es una máquina sin utilidad, necesita de programas que le permitan ejecutar aplicaciones, además es un sistema complejo compuesto de varios elementos. La gestión de estos elementos es una labor complicada. Si la gestión la tienen que hacer los propios programas de aplicación, que además pueden estar ejecutándose simultáneamente, es muy probable que los programadores se vieran desbordados por las dificultades. Hace tiempo que se vio la necesidad de distinguir dos tipos de programas los de sistemas y los de aplicación.

En un sentido amplio los programas del sistema se encargan de controlar las operaciones propias del computador, mientras que los de aplicación son los que resuelven problemas específicos a los usuarios. De los programas del sistema lo más importante es el sistema operativo cuyo objetivo es que el computador se pueda utilizar de una manera cómoda y eficiente, proporcionando un caparazón a la máquina desnuda que permite dar la visión de una máquina virtual, con la que es factible comunicarse al crearse un interfaz entre el usuario y la máquina y gestionar los recursos del sistema.

El sistema operativo oculta la complejidad del hardware y proporciona un interfaz más adecuado para usar el sistema. Actúa como mediador, de forma que los programadores y los programas de aplicación puedan acceder y utilizar los recursos y servicios del sistema de una forma más fácil, sin preocuparse de la gestión de los mismos.

¿Que es un sistema operativo?.

El Sistema Operativo se coloca una capa por encima del hardware, de forma que lo oculta al resto de los programas y a los usuarios. Desde esta perspectiva el sistema operativo no es más que un programa que controla la ejecución de los programas de aplicaciones y actúa como un interfaz entre los usuarios del computador y el hardware. Sus objetivos son:

- Comodidad para los usuarios. El Sistema Operativo hace que el computador sea más fácil de utilizar. En este caso hace que los usuarios vean una máquina virtual o extendida, que es más sencilla de programar y de utilizar que la máquina desnuda.
- Eficiencia. El Sistema Operativo gestiona los recursos del sistema de forma más eficaz. Su función es la de un gestor de recursos.

Los Sistemas Operativos como máquinas virtuales.

La utilización del hardware es difícil, sobre todo en operaciones de E/S; el programador y usuario, no desea enfrentarse a toda esa problemática, sino que desea una abstracción sencilla y fácil de entender. Para ocultar esta problemática está el Sistema Operativo, que permite ver una abstracción del hardware que lo presenta como una máquina virtual que entiende ordenes de un nivel superior. En este sentido, el Sistema Operativo tiene que proporcionar servicios para las funciones siguientes:

Creación de programas.
Ejecución de programas.
Operaciones de Entrada/Salida
Manipulación y control de archivos.
Detección de errores.
Control de acceso al sistema.
Elaboración de informes estadísticos.

El Sistema Operativo como gestor de recursos

La visión anterior corresponde de abajo hacia arriba. Como alternativa se puede considerar el Sistema Operativo como el gestor de todos los elementos que componen el sistema. En este caso su función es proporcionar a los programas que compiten por ellos una asignación ordenada de los procesadores, memoria y, los periféricos.

El Sistema Operativo realiza este control de una forma peculiar, ya que funciona igual que otro programa. Otra característica que lo hace peculiar es que pierde el control del procesador y debe esperar para disponer de él. Desde este punto de vista el Sistema Operativo no es más que otro programa del computador que el procesador no distingue del resto. La gran diferencia es la finalidad, ya que la UCP dedicará parte de su tiempo a ejecutar otros programas según lo planifique el Sistema Operativo y también control el acceso a los otros recursos del sistema, aunque esto le haga perder momentáneamente el control del procesador.

Conceptos fundamentales.

Procesos.

Es un término más general que trabajo (job). Básicamente un proceso es un programa en ejecución, de una manera más formal se puede definir como “la entidad que puede ser asignada a un procesador y ejecutada por él”.

En un sistema de tiempo compartido cada cierto tiempo el Sistema Operativo tiene que parar el proceso en ejecución y arrancar otro, de forma que cada proceso tenga asignado el procesador durante un intervalo de tiempo prefijado. En esta situación, el proceso que se ha detenido se dice que se ha suspendido y será arrancado posteriormente en el mismo estado en el que se suspendió. La información sobre cada proceso se almacena en una tabla de procesos construida y mantenida por el

propio Sistema Operativo cada entrada a la tabla incluye un puntero a la ubicación del bloque de memoria que contiene al proceso, también puede contener parte o todo el contexto de ejecución del proceso.

Gestión de memoria y del sistema de archivos

Los programadores y usuarios deben disponer de un soporte para la programación modular y para un uso flexible de los datos. Además el administrador del sistema necesita un control ordenado y eficiente del almacenamiento asignado. Para satisfacer estos requerimientos el Sistema Operativo tiene que responsabilizarse de:

1. Aislar los procesos, de manera que no interfieran en los datos o memoria de otro.
2. Ubicar y gestionar automáticamente a los procesos, de manera que sean transparentes a los programadores. El sistema asignará memoria a los procesos según las limitaciones que tengan y teniendo en cuenta las necesidades de los distintos trabajos.
3. Soportar una programación modular, de forma que se puedan definir módulos de programas en los que se puedan alterar dinámicamente sus tamaños.
4. Controlar el acceso y proteger la memoria. Aunque se comparte la memoria, un programa no debe acceder al espacio de direccionamiento de otro. Hay ciertas partes de memoria que puede acceder los usuarios de manera controlada.
5. Disponer de un medio de almacenamiento de larga duración.

Para cumplir estas responsabilidades el Sistema Operativo cuenta con la noción de memoria virtual y las funciones del sistema de archivos. Con el concepto de memoria virtual los programas pueden direccionar la memoria desde un punto de vista lógico, sin considerar la cantidad de memoria realmente instalada en el sistema, cuando un programa se ejecuta, sólo se carga en memoria principal la parte de código y datos que se van a utilizar. Para un almacenamiento de larga duración se construye el sistema de archivos. La información se almacena en archivos y estos se mantienen en directorios, que pueden tener una estructura arborescente dentro de otros directorios.

Llamadas al sistema.

Este concepto corresponde a la interfaz entre el Sistema Operativo y los programas y usuarios. Las llamadas al sistema se pueden hacer de varias formas dependiendo del computador. Para hacer la llamada se requiere cierta información, a parte de la identidad de la llamada, esta información depende del Sistema Operativo y de la llamada en concreto. Las llamadas al sistema se pueden agrupar en cinco categorías:

- Control de procesos.
- Manipulación de archivos.
- Manipulación de periféricos.
- Mantenimiento de la información
- Comunicaciones.

Gestión y planificación de los recursos.

Entre las tareas claves de un Sistema Operativo está la de gestionar los diferentes recursos del sistema y planificar la utilización de los mismos de forma más eficiente por los procesos en ejecución. La política de planificación y asignación de recursos tiene que ser justa y eficiente. Por otro lado existen trabajos con diferentes exigencias de servicio. El Sistema Operativo debe intentar hacer la planificación y la asignación considerando el total de los requerimientos.

Protección y seguridad de la información.

Estructura de los Sistemas Operativos.

Los Sistema Operativo forman un sistema vasto y complejo, que sólo se puede construir si se particiona en computadores más pequeños. Aunque no todos los Sistema Operativo tienen la misma estructura, es usual dividirlos en el siguiente conjunto de componentes básicos.

- Gestor de procesos
- Gestor de la Memoria Principal.
- Gestor de almacenamiento secundario y del sistema de archivos.
- Gestor del sistema de Entrada/Salida.
- Sistema de protección
- Sistema de comunicación.
- Intérprete de órdenes.

La estructura jerárquica de un Sistema Operativo moderno diferencia sus funciones de acuerdo a su complejidad, sus características en el tiempo y su nivel de abstracción. En este caso se puede ver el sistema como una serie de niveles. Cada uno implementa un subconjunto de las funciones requeridas por el Sistema Operativo y a su vez dependen del nivel inmediato inferior para realizar funciones más elementales y ocultar sus detalles a niveles superiores a los que proporcionan sus

servicios. Los niveles se tiene que definir de forma que los cambios que se hagan en un nivel no supongan modificaciones en los otros.

Diseño e implementación de Sistemas Operativos.

Cuando se va a diseñar un Sistema Operativo, el primer problema es definir adecuadamente sus objetivos y especificaciones. Hay que tener en cuenta que se verá afectado por el hardware elegido y por el tipo de sistema que se quiere implementar. Los requisitos a especificar se dividen en dos grandes grupos: requisitos del usuario y requisitos del sistema. También se debe tener en cuenta otra exigencia del sistema, su capacidad de evolución, de forma que se le permita la introducción de nuevas funciones sin interferir con los servicios ya disponibles. Las razones que pueden motivar la evolución son:

- Actualizaciones de hardware y aparición de nuevos tipo de máquina.
- Posibilidad de añadir nuevos servicios solicitados por los usuarios.
- Necesidad de corregir fallos.

Inicialmente los Sistemas Operativos se codificaban en ensamblador, actualmente lo más normal es escribirlos en un lenguaje de alto nivel, y las partes muy dependientes del hardware en ensamblador. Las ventajas de este enfoque son:

- El código se puede escribir más rápidamente.
- Facilidad de comprender y depurar el código.
- Transportabilidad del Sistema Operativo a otras plataformas.

Como desventaja tiene una menor optimización de velocidad y almacenamiento.

Conclusiones

Lo que hace que un computador sea una máquina útil es el software y el programa más importante es el Sistema Operativo. Sus principales funciones y objetivos son:

- Proporcionar una máquina virtual a los usuarios y a las aplicaciones, que resulte más fácil de programas y de utilizar.
- Ser un gestor de los recursos de la máquina

Los conceptos incorporados durante la evolución de los Sistemas Operativos son:

- Procesos
- Gestor de memoria
- Sistema de archivos.
- Llamadas al sistema
- Gestión y planificación de los recursos.
- Protección y seguridad de la información.

Los Sistemas Operativos considerados como un programa han alcanzado un tamaño muy grande, debido a que tienen que hacer muchas cosas. Por esta razón para construir un Sistema Operativo es conveniente dividirlo en componentes más pequeñas y hacer una estructura en niveles.

2-. PROCESOS

Introducción a los procesos.

Un Sistema Operativo se define como “un conjunto de extensiones software del hardware original, que culminan en una máquina virtual que sirve como un entorno de programación de alto nivel que gestiona el flujo de trabajo en una red de computadores”. El Sistema Operativo proporciona una serie de servicios y una interfaz a los mismos.

En los primeros ordenadores sólo se ejecutaba un programa o trabajo a la vez, sin embargo en seguida se puso de manifiesto que algunos programas estaban limitados en su velocidad de ejecución por el uso de unidades de E/S, y otros que hacían poco uso de ellas, ninguno de los dos hacía un uso total del computador, este problema se intentó resolver con la multiprogramación, que con el entrelazado y solapamiento de la ejecución de un programa, se intenta mantener todos los recursos del sistema de la forma más ocupada. Los sistemas con multiprogramación interactiva, en los que se asigna a cada programa un mismo intervalo de tiempo de ejecución se denominan sistemas de tiempo compartido.

Se utiliza el término multitarea para referirse a la capacidad que tienen los Sistema Operativo de ejecutar de forma simultánea varios procesos, reservándose el término multiprocesamiento a un computador que dispone de varios procesadores. Un programa se ejecuta sólo una vez, si es usado por varios usuarios generará diferentes procesos que compiten por los recursos del sistema, por lo que precisan de herramientas de sincronización y comunicación entre los mismos.

Relación entre procesos.

El Sistema Operativo debe suministrar los servicios para la realización de las siguientes actividades:

- Ejecución concurrente de los procesos.
- Sincronización entre procesos.
- Comunicación entre procesos.

Además el Sistema Operativo debe disponer de algoritmos de gestión y planificación de procesos que se encarguen de las siguientes acciones:

- Decidir qué proceso se ejecutará o tomará el procesador.
- Llevar la cuenta de los estados de los procesos.

Dependiendo de la interacción de los procesos, éstos se clasifican en independientes, cooperativos y competitivos. Los procesos independientes no se comunican o sincronizan entre ellos. Si el sistema sólo tiene un procesador no existen ya que todos compiten por el uso del mismo y probablemente de la memoria y dispositivos de E/S. Los procesos cooperativos se comunican y sincronizan sus actividades para realizar una labor común. Cuando se comparten recursos los procesos compiten entre ellos, para conseguir un acceso ordenado a estos recursos se necesita la sincronización y a veces la comunicación entre los procesos.

Especificación de los procesos.

Los procesos generados por el Sistema Operativo se denominan implícitos. Una vez terminada la ejecución de los mismos, su eliminación también la realiza el propio Sistema Operativo. Así mismo el Sistema Operativo también proporciona en tiempo real los servicios que son necesarios para que el usuario pueda definir procesos de forma explícita. Los programas acceden a estos servicios realizando llamadas al sistema.

Al comienzo de la ejecución del programa principal de un usuario se inicia la ejecución de un proceso. A su vez el proceso puede crear otros procesos. El proceso que crea otro proceso se denomina padre (parent process); y el proceso creado se denominan hijo (child process). Una vez creado el hijo la ejecución de ambos transcurre de manera concurrente. Se puede crear una jerarquía arborescente de procesos, en la que un padre puede tener varios hijos y éstos pueden tener otros hijos, etc. pero donde un hijo sólo puede tener un padre.

- *Creación de procesos.*

Función create: Crea un proceso. Devuelve un número entero que identifica de forma única el proceso creado, se denomina *pid* o *id* del proceso. El proceso queda en estado suspendido.

Función resume: Recibe como parámetro el *pid* y pasa el proceso al estado preparado.

Kill: Permite terminar, eliminar el proceso.

Estado de los procesos.

Activo (Run): Se está ejecutando en un instante dado.

Preparado (Ready): Listo para ejecutarse, espera que el/un procesador quede libre.

Bloqueado (Block) o suspendido: A la espera de que se cumpla alguna condición.

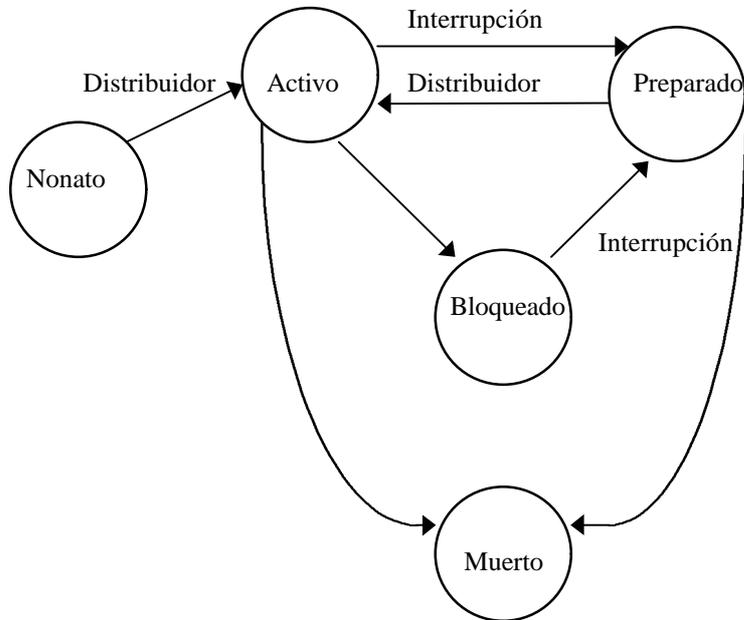
Muerto: Ha terminado su ejecución o el SO tiene un fallo o detecta un error fatal y lo transfiere a este estado.

Nonato: El programa existe pero no es conocido por el Sistema Operativo.

Se denomina estado global del sistema en un instante determinado, al conjunto de recursos y procesos existentes con sus estados correspondientes. El estado global es cambiado por el Sistema Operativo modificando el estado de los procesos y la asignación de recursos en respuesta a eventos externos o internos.

Transición entre los estados.

El Sistema Operativo dispone de un módulo, el distribuidor (dispatcher) que se encarga de activar los procesos que están en el estado preparado de acuerdo con unos criterios determinados. Toda interrupción hace que la tarea que está activa deje de ejecutarse, el Sistema Operativo decidirá entre los procesos que están preparados cuál de ellos tiene prioridad para pasar al estado activo.



Bloque de control de procesos (BCP).

La creación de un proceso origina la creación de su Bloque de Control de Procesos (BCP) que sirve para describirlo hasta que se elimina o pasa al estado de muerto, donde no posee BCP. El Bloque de Control de Proceso guarda la información que necesita el sistema para controlar al proceso y dar cuenta de sus recursos y toda la que influye en la ejecución de un programa.

- Identificador único del proceso (pid)
- Estado del proceso
- Prioridad
- Estado hardware (Contador de programa, puntero de pila ...)
- Información de gestión de memoria (punteros ,tablas, ...)
- Información de estado del sistema de E/S.
- Información de contabilidad y planificación.

Listas de procesos.

El Sistema Operativo mantiene listas de Bloque de Control de procesos para cada uno de los estados del sistema. Cada proceso pertenece a una única lista. Posee una lista de procesos preparados, una lista de los bloqueados y en el caso de sistemas multiprocesadores una lista de procesos activos.

Procesos y hebras.

La ejecución y posesión de recursos pueden verse como dos conceptos independientes y que pueden tratarse de forma separada. De esta manera, se puede describir el proceso como una entidad formada por una o más unidades de ejecución denominadas hebras (threads) y un conjunto de recursos asociados. A las hebras se les llama procesos primitivos. Cuando un proceso pasa al estado activo, una hebra toma posesión del procesador; ésta se puede interrumpir para pasar a ejecutar otra hebra, y así sucesivamente hasta que el proceso sale de dicho estado. Cada hebra posee su propio vector de estado. Las hebras sólo pueden existir fuera de un proceso y sólo pueden pertenecer a uno de ellos.

Los recursos no están asociados a las hebras sino al procesos; éste es el que puede acceder de forma protegida a los recursos del sistema y se encarga de su gestión entre las hebras. El proceso se puede ver como un espacio de dirección virtual que contiene la imagen de un programa y es la unidad propietaria de los recursos para un conjunto de hebras. El proceso sirve así para la ejecución de las hebras. Con esta concepción se mejora el SO ya que la creación de una hebra dentro de un proceso

requiere menos tiempo que la creación de un proceso nuevo, ya que su estado se ve reducido con respecto al del proceso. También mejora el tiempo de conmutación entre hebras con respecto al de los procesos.

El planificador de procesos.

El planificador es el software del SO encargado de asignar los recursos de un sistema entre procesos que lo solicitan, decidiendo quien lo recibirá. Hay dos tipos de planificadores que coexisten en el SO: el planificador a largo plazo (PLP) y el planificador a corto plazo (PCP):

El planificador a largo plazo, o planificador de trabajos, determina qué trabajos se admiten en el sistema para su procesamiento y son, por lo tanto, cargados en la memoria disponible. En los sistemas por lotes se presentan más trabajos de los que se pueden admitir para ejecutar inmediatamente, estos trabajos se guardan en un dispositivo de almacenamiento para su posterior ejecución. El PLP se encarga de seleccionar los trabajos almacenados de acuerdo con una cierta política de gestión. El PLP debe realizar una mezcla adecuada de trabajos destinados al procesador y al sistema de E/S con el fin de mantener un uso adecuado de los recursos.

El planificador a corto plazo (PCP) o planificador del procesador, selecciona al proceso que pasará al estado activo de entre todos los procesos residentes en memoria que están en estado preparado, es un algoritmo sencillo ya que su uso es muy frecuente. Los eventos que producen una invocación al PCP son:

- Señales de reloj del sistema.
- Interrupciones.
- Finalización de las operaciones E/S.
- Llamadas al sistema Operativo.
- Envío y recepción de señales.
- Activación de programas interactivos.

El planificador a medio plazo (PMP) se basa en el hecho de que es conveniente llevar a la memoria secundaria un proceso que ha sido suspendido en su ejecución por algún evento, lo que permite liberar la memoria principal.

Criterios para la planificación.

Eficacia: Porcentaje del tiempo medio de utilización.

Rendimiento (Throughput): Número de procesos completados por unidad de tiempo.

Tiempo de regreso (Turnaround): Tiempo desde que un proceso se crea hasta que se completa por el sistema.

Tiempo de espera: Tiempo de espera de un proceso hasta que se le concede el procesador.

Tiempo de respuesta a un evento: Intervalo de tiempo desde que se señala un evento hasta que se ejecuta la primera instrucción de su rutina de servicio.

Eficiencia: Tcpu/Tretorno.

Planificación por expropiación:

Los procesos con más prioridad reemplazan a cualquier tarea que tenga menor prioridad. Se activa el PCP cada vez que se produce un evento que origina un cambio en el estado general del sistema. En los algoritmos con estrategia de *no expropiación* el proceso que está activo permanece en el procesador hasta que él mismo devuelve el control al SO.

Planificación FCFS. Primero en llegar, primero en ser servido.

Es del tipo no expropiativo, se emplea dentro de otros esquemas. Los procesos en el estado preparado acceden al procesador en el orden que llegan a dicho estado.

Planificación SJF. Primera tarea más corta.

Estrategia de planificación no expropiativa en la que a cada trabajo se le asocia un tiempo de finalización de ejecución, realizándose la selección en base a este tiempo. Es una estrategia óptima para minimizar el tiempo medio de espera. Su dificultad es la estimación del tiempo restante de ejecución.

Planificación SRT. Tiempo más corto que queda.

Es una versión expropiativa de SJF.- Se elige aquel proceso al que le queda menos tiempo para terminar su ejecución, incluyendo los nuevos que lleguen. Frente al SJF tiene mayor frecuencia de invocación al planificador y una carga superior en labores, lo que puede desembocar en un menor rendimiento del procesador.

Planificación RR. Rotación Circular (Round Robin).

A todos los procesos del estado preparado se les asigna un tiempo de ejecución denominado *cuanto*. El planificador asigna el procesador secuencialmente a cada tarea. Es un método muy adecuado en sistemas de tiempo compartido. La realización de la planificación RR necesita de un temporizador que controle el cuanto asignado a los procesos y genere una interrupción siempre que finalice. La interrupción hace que se llame al PCP que guarda el contexto de ejecución. El problema de RR es fijar el tamaño del cuanto, ya que supone un compromiso entre la eficacia del procesador y el tiempo de respuesta.

El Reloj de Tiempo Real

Además del reloj que control la velocidad del procesador la mayoría de sistemas dispone de un reloj de tiempo real (RTR) que se encarga de generar una señal de forma periódica utilizada para producir una interrupción a un intervalo de tiempo fijo. La interrupción debe ser atendida rápidamente para evitar que se pierda.

Puede ser conveniente ralentizar el reloj, con el fin de lograr una funcionalidad más rica. La velocidad real de ejecución efectiva del código de tratamiento se le conoce como velocidad de *tic*. EL sistema operativo utiliza el RTR. Para:

Limitar el tiempo que un proceso puede estar en ejecución.

Proporcionar servicios al usuario.

La rutina de tratamiento de la interrupción del RTR se encarga de mirar la lista de los procesos retardados y de pasar al estado preparado a aquellos para los que ha transcurrido el tiempo de retardo.

Planificación por prioridades.

En los algoritmos de planificación por prioridad cada proceso tiene asignada una y el de mayor prioridad es el que toma el procesador. La asignación puede ser estadística o dinámica. Si es estadística no cambia durante el tiempo en que dura el proceso. En el caso de la asignación dinámica la prioridad puede ser modificada por el usuario o el sistema. Los algoritmos con planificación de prioridades pueden ser de tipo expropiación o no expropiación. En los primeros, si un proceso pasa al estado preparado y tiene una prioridad mayor que el que se ejecuta, el PCP llevará a ejecución el proceso preparado. En una estrategia de no expropiación el proceso aunque tenga mayor prioridad deberá esperar que acabe el que se está procesando. Los algoritmos con expropiación y con prioridades se dice que están guiados por eventos. Cuando el algoritmo permite aumentar la prioridad de los que llevan un tiempo de espera elevado se conoce como prioridad por envejecimiento (aging).

Planificación MLQ. Colas multinivel..

Se crean colas de tareas separadas que se gestionan por criterios diferentes. Cada tarea se asigna a una sola cola de acuerdo a alguna propiedad de la tarea. Debe existir un criterio de planificación entre colas (suele ser el de prioridad fija con expropiación). Otra posibilidad de gestión de las colas es la de compartir el tiempo entre las colas: a cada una de ellas se le asigna una porción de tiempo que debe de utilizar para planificar entre las distintas tareas de su cola.

Planificación MLFQ Colas multinivel con realimentación.

Para permitir la movilidad entre las colas, estas se organizan según sus características de uso del procesador. Las tareas que emplean poco el procesador se mantienen en las colas de mayor prioridad y las que lo utilizan mucho se sitúan en las colas de menor prioridad. El movimiento de las tareas entre las colas se realiza según su comportamiento dinámico a lo largo de su tiempo de vida efectiva. Existen muchos algoritmos de planificación por MLFQ que se pueden definir con los siguientes parámetros:

- Número de colas.
- Algoritmo de planificación de cada cola.
- Métodos que determinan el movimiento de las tareas entre colas.
- Métodos que indica la cola en la que entre una tarea cuando necesita un servicio.

Planificación para sistemas de tiempo real.

Un Sistema de Tiempo Real (STR) es un sistema informático que responde a estímulos producidos externamente en un intervalo especificado de tiempo finito. Las tareas se suelen dividir en tres niveles de prioridad.

Nivel de interrupción: Respuesta muy rápida. Tareas del RTR y distribuidor de nivel de reloj. Tareas organizadas por prioridades.

Nivel de reloj: Tareas que requieren un procesamiento repetitivo y una sincronización previa.

Nivel base: Tareas que no tienen fecha límite para realizarse, permiten error en la sincronización. Su planificación suele ser RR o prioridades fijas o variables.

3- Sincronización y Comunicación de Procesos

Exclusión mutua.

El método más sencillo de comunicación entre los procesos de un programa concurrente es el uso común de unas variables de datos. El problema de este sistema es que la acción de un proceso interfiere en las acciones de otro de una forma no adecuada. Para evitar este tipo de errores se pueden identificar aquellas regiones de los procesos que acceden a variables compartidas y dotarlas de la posibilidad de ejecución como si fueran una única instrucción. Se denomina sección crítica a aquellas partes de los procesos concurrentes que no pueden ejecutarse de forma concurrente o, que desde otro proceso se ven como si fueran una única instrucción. Esto quiere decir que si un proceso entra a ejecutar una sección crítica en la que accede a unas variables compartidas, entonces otro proceso no puede entrar a ejecutar una región crítica en la que se modifique las variables compartidas con el anterior. Las secciones críticas se pueden agrupar en clases, siendo mutuamente exclusivas las secciones críticas de cada una. Para conseguir dicha exclusión se deben implementar protocolos software que impidan o bloqueen el acceso a una sección crítica mientras está siendo utilizada por un proceso.

Bloqueo mediante el uso de variables compartidas.

Se usa una variable de tipo booleano llamada "flag" que es compartida por los procesos, se asocia a cada recurso que se comparte. Si el sistema dispone de una instrucción que permita comprobar el estado de la flag y modificarlo simultáneamente, el programa permitiría el uso del recurso sin entrelazamiento. En caso de usar dos indicadores para un recurso, puede ocurrir que ambos procesos vean los indicadores contrarios como ocupados, y permanezcan a la espera de liberación del recurso, pero esto no puede ocurrir al no poder entrar ninguno en su sección crítica (Interbloqueo o deadlock).

Algoritmo de Peterson:

Una posible solución es introducir una variable adicional, denominada *turno* (Peterson, 1981), que sólo resulta útil cuando se produzca un problema de petición simultánea de acceso a la región crítica:

(* Exclusión mutua; Solución de Peterson *)

```

module Exclusion_ Mutua_ P;
var flag1, flag2: Boolean;
    turno : integer;
Procedure bloqueo (var mi_flag, su_flag: boolean; su_turno:integer);
begin
    mi_flag := true;
    turno := su_turno;
    while su_flag and (turno := su_turno) do; end
end bloqueo;
Procedure desbloqueo (var mi_falg: boolean);
begin
    mi_flag := false;
end desbloqueo;
process P1
begin
    loop
        bloqueo(flag1, flag2, 2);
        (* Uso del recurso Sección Crítica *)
        desbloqueo (flag1);
        (* Resto del proceso *)
    end;
end P1;
process P2
begin
    loop
        bloqueo(flag2, flag1, 2);
        (* Uso del recurso Sección Crítica *)
        desbloqueo (flag2);
        (* Resto del proceso *)
    end;
end P2;

```

```

begin (* Exclusión_ Mutua_ P *)
flag1 := false
flag2 := false;
cobegin
    P1;
    P2;
coend
end Exclusión_ Mutua_ P.

```

Si sólo uno de los procesos intenta acceder a la sección crítica lo podrá hacer sin ningún problema. Sin embargo, si ambos intentan entrar a la vez, el valor del turno se pondrá a 1 y 2 pero sólo un valor de ellos permanecerá al escribirse sobre el otro, permitiendo el acceso de un proceso a su región crítica. El algoritmo permite resolver el problema de exclusión mutua y garantiza que ambos procesos usarán de forma consecutiva el recurso en el caso de que lo utilicen a la vez y se impedirá el cierre del otro proceso.

Algoritmo de Dekker.

Utiliza también una variable de turno, que sirve para establecer la prioridad relativa de los dos procesos y actúa en la sección crítica, lo que evita posibles interferencias entre procesos:

(*Exclusión mutua: Solución de Dekker *)

```

module Exclusion_Mutua_D;
var flag1, flag2: Boolean;
    turno : integer;
Procedure bloqueo (var mi_flag, su_flag: boolean; su_turno:integer);

begin
    mi_flag := true;
    while su_flag do(* otro proceso en la sección crítica*)
        if turno = su_turno then
            mi_flag := false;
            while turno = su_turno do
                ; (*espera que el otro acabe*)
            end;
            mi_flag := true;
        end;
    end;
end bloqueo;
Procedure desbloqueo (var mi_falg: boolean; su_turno :integer);

begin
    turno := su_turno
    mi_flag := false;
end desbloqueo;
process P1
begin
    loop
        bloqueo(flag1, flag2, 2);
        (* Uso del recurso Sección Crítica *)
        desbloqueo (flag1);
        (* Resto del proceso *)
    end;
end P1;
process P2
begin
    loop
        bloqueo(flag2, flag1, 2);
        (* Uso del recurso Sección Crítica *)
        desbloqueo (flag2);
        (* Resto del proceso *)
    end;
end P2;

```

```

begin (* Exclusión_Mutua_D *)
flag1 := false
flag2 := false;
cobegin
    P1;
    P2;
coend
end Exclusión_Mutua_D.
    
```

El programa se inicia con el valor de turno 1, lo que da prioridad al proceso P1. Si ambos programas piden a la vez el acceso a su sección crítica, activan sus respectivos indicadores y comprueban si el indicador del otro está activado. Ambos encuentran que sí, por lo que pasan a evaluar el turno. El segundo encuentra que no es su turno, desactiva su indicador y se queda en espera de que lo sea. P1 comprueba que si es su turno, entra en su región crítica y dará el turno a P2 antes de desactivar su indicador.

Los algoritmos de Peterson y Dekker se pueden extender al caso más general con n procesos en ejecución concurrente; pero no son soluciones adecuadas ya que la espera de acceso a un recurso siempre se realiza de forma "ocupada". El proceso se queda permanente comprobando una variable, lo que puede suponer un derroche de los recursos del sistema.

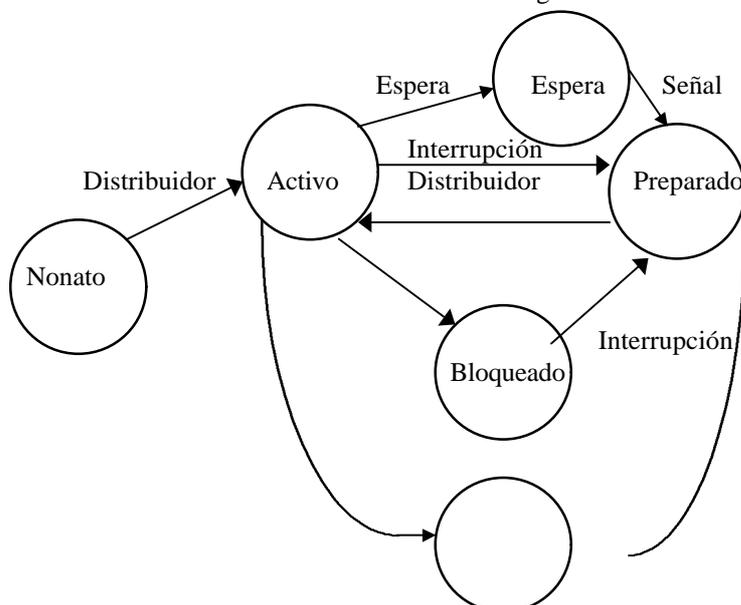
Semáforos.

Es una solución sencilla y elegante del problema de la exclusión mutua. Esta técnica permite solucionar la mayoría de los problemas de sincronización entre procesos. Un semáforo binario es un indicador de condición (s) que registra si un registro está disponible o no. Un semáforo sólo puede tomar dos valores; 0 y 1. Si para un semáforo, $S=1$ el recurso está disponible y la tarea lo puede utilizar; si $S=0$ el recurso no está disponible y el proceso debe esperar.

Los semáforos se implementan con una cola de tareas a la que se añaden los procesos que están en espera del recurso. Solo se permiten tres operaciones sobre un semáforo:

<p>Inicializa (s: Semáforo_Binario; v: integer) poner el valor del semáforo s al valor de v (0,1)</p>
<p>Espera(s) if s = 1 then s := 0 else Suspender la tarea que hace la llamada y ponerla en la cola de tareas.</p>
<p>Señal (s) if cola de tareas vacía then s := 1 else Reanudar la primera tarea de la cola tareas.</p>

Estas operaciones son procedimientos que se implementan como acciones indivisibles. En sistemas con un único procesador bastará simplemente con inhibir las interrupciones durante la ejecución de las operaciones del semáforo. Al introducir el semáforo se crea un nuevo estado en el diagrama de transiciones, el de espera.



Muerto ←

Exclusión mutua con semáforos.

La exclusión mutua se realiza fácilmente usando semáforos. La operación de espera se usará como procedimiento de bloqueo antes de acceder a una sección crítica y la operación señal como procedimiento de desbloqueo. Se emplean tantos semáforos como clases de secciones críticas se establecen.

```

Process P1
begin
    loop
        Espera (s)
        Sección crítica
        Señal (s)
        (* Resto del proceso *)
    end;
end P1.

```

Sincronización:

Las operaciones espera y señal no se utilizan dentro de un mismo proceso sino que se dan en dos procesos separados, el que ejecuta *espera* queda bloqueado hasta que el otro ejecuta *señal*.

```

Module Sincronización
    var sincro: Semáforo
    process P1 (*proceso que espera *)
    begin
        .....
        espera (sincro)
        .....
    end
    process P2 (*proceso que señala *)
    begin
        .....
        señal (sincro)
        .....
    end
    begin (*sincronización *)
        inicializa (sincro, 0)
    Cobegin
        P1
        P2
    Coend
end Sincronización.

```

El semáforo binario se inicializa a cero de modo que cuando el proceso P1 ejecuta la operación de espera se suspende hasta que el proceso P2 ejecuta la operación señal.

El problema del productor - consumidor aparece en distintos lugares del sistema operativo y caracteriza las situaciones en las que existen un conjunto de procesos que producen información que otros procesos consumen, siendo diferentes las velocidades de producción y consumo de la información, lo que hace necesario que se establezca una sincronización de los procesos.

(* Problema del Productor - Consumidos solucionado con semáforos *)

```

module Productor_Consumidor
    var
        BufferComun : buffer;
        AccesoBufer,
        Nolleno, Novacio: semáforo;
    process Productor;
    var x : dato;
    begin
        loop
            produce (x)

```

```

    espera (AccesoBuffer);
if Lleno then
    señal (AccesoBuffer);
    espera (Nolleno);
    espera (AccesoBuffer);
end
Poner (x)
señal (AccesoBuffer);
señal (Novacio);
end
end Productor.

process Consumidor;
var x : dato;
begin
loop
    espera (AccesoBuffer);
if Vacío then
    señal (AccesoBuffer);
    espera (NoVacio);
    espera (AccesoBuffer);

end
Tomar (x)
señal (AccesoBuffer);
señal (Nolleno);
Consume (x);
end
end Consumidor.

Begin
inicializa (AccesoBuffer,1)
inicializa (Nolleno,1);
inicializa (Novacio,0);
Cobegin
    Productor;
    Consumidor;
Coend
end Productor_ Consumidor.

```

Versión general de los semáforos.

El semáforo binario resulta adecuado cuando hay que proteger un recurso que pueden compartir varios procesos, pero cuando lo que hay que proteger es un conjunto de recursos similares, se usa una versión más general que lleve la cuenta del número de recursos disponibles (n). Al igual que en los semáforos binarios las operaciones son indivisibles.

<p>Inicializa (s: Semáforo_Binario; v: integer)</p> <p>poner el valor del semáforo s al valor de v (n) número de suspendidos := 0</p>
<p>Espera(s)</p> <p>if s < 0 then s := s - 1 else número de suspendidos := número de suspendidos + 1 Suspendir la tarea que hace la llamada y ponerla en la cola de tareas.</p>
<p>Señal (s)</p> <p>if número de suspendidos > 0 then número de suspendidos := número de suspendidos - 1 pasar al estados preparado un proceso suspendido else s := s + 1.</p>

Monitores.

Un monitor es un conjunto de procedimientos que proporciona el acceso con exclusión mutua a un recursos o conjunto de recursos compartidos por un grupo de procesos. Los procedimientos van encapsulados dentro de un módulo que tiene la propiedad especial de que sólo un proceso puede estar activo cada vez para ejecutar un procedimiento del monitor.

La ventaja que presenta un monitor en la exclusión mutua frente a los semáforos, es que está implícita: la única acción que debe realizar el programador del proceso que usa un recurso es invocar una entrada del monitor. Si el monitor se ha codificado correctamente no puede utilizarse de forma incorrecta por un programa de aplicación, lo que no ocurre con los semáforos.

Los monitores, por si mismos, no proporcionan un mecanismo para la sincronización de tareas debiéndose utilizar para ello señales. Una variable que se utilice como mecanismo de sincronización en un monitor se conoce como *variable de condición*. A cada causa diferente por la que un proceso debe esperar se le asocia una variable de condición. Sobre ellas sólo se puede actuar con dos procedimientos: *espera* y *señal*. Cuando un procedimiento efectúa una operación de espera se suspende y se coloca en una cola asociada a dicha variable de condición. La diferencia con el semáforo es que la ejecución de esta operación suspende el proceso que la emite, lo que implica la liberación del monitor permitiendo que entre otro proceso.

Sintaxis del monitor.

```

Monitor: nombre_monitor;
           declaración de tipos y procedimientos que se importan y exportan

           declaración de las variables locales del monitor
           y de las variables de condición.
Procedure Proc1 (. . .)
begin . . . end;

           Procedure Proc2 (. . .)
begin . . . end;
           . . .
           Procedure ProcM (. . .)
begin . . . end;

begin
           Inicialización del monitor
end

```

El monitor no tiene por qué exportar todos su procedimientos sino sólo aquellos que sean *públicos*, manteniendo como *privados* aquellos a los que sólo pueden acceder otros procedimientos del monitor que se exportan y actúan como puertas de entrada al monitor.

```

Monitor : semáforo;
from condiciones import condición, espera, señal;
export sespera, sseñal;

var    ocupado: boolean;
        libre: condición;

procedure sespera (var libre: condición);
begin
        if ocupado then
            espera (libre);
            ocupado := true;
        end
end sespera
procedure sseñal (var libre: condición);
begin
        ocupado := false;
        señal (libre);
end sseñal
begin
        ocupado := false

```

end

```

Monitor :productor_ consumidor;
from condiciones import condición, espera, señal;
export poner, tomar;

const tamaño = 32;
var buff: array [0 .. tamaño-1] of dato
    cima, base : 0 .. tamaño -1;
    espaciadisponible, itemdisponible : condición;
    numeroenbuffer : integer;

procedure poner (item: dato);
    begin
        if numeroenbuffer = tamaño then
            espera (espaciadisponible);
        end
        buff[cima] := item;
        numeroenbuffer := numeroenbuffer + 1;
        cima := (cima +1) mod tamaño;
        señal (itemdisponible)

    end poner
procedure tomar (item: dato);
    begin
        if numeroenbuffer = 0 then
            espera (itemdisponible);
        end
        item := buff[base];
        numeroenbuffer := numeroenbuffer - 1;
        base := (base +1) mod tamaño;
        señal (espaciadisponible)

    end tomar;

begin (* inicialización *)
    numeroenbuffer := 0;
    cima := 0;
    base := 0;
end.

```

Un proceso que desee poner un dato en el buffer deberá esperar al monitor en la cola de entrada si éste está ocupado. Una vez dentro, si el buffer está lleno se suspenderá cuando ejecute *espera (espacio disponible)*, colocándose en la cola de esta variable condición.

Mensajes.

Los mensajes proporcionan una solución al problema de la concurrencia de procesos que integra la sincronización y la comunicación entre ellos y resulta adecuado tanto para sistemas centralizados como para distribuidos. La comunicación mediante mensajes necesita siempre un proceso *emisor* y uno *receptor* y la información debe intercambiarse. Las operaciones básicas para una comunicación mediante mensajes que proporciona todo Sistema Operativo son **enviar** (mensaje) y **recibir** (mensaje). Las acciones de transmisión de información y de sincronización se ven como actividades inseparables.

La comunicación por mensajes requiere que se establezca un enlace entre el receptor y el emisor, la forma del mismo puede variar mucho de sistema en sistema, su implementación varía dependiendo de los aspectos siguientes:

- Modo de nombrar los procesos.
- Modelo de sincronización.
- Almacenamiento y estructura del mensaje.

Modos de nombrar los mensajes.

• Ambos procesos, el que envía y el que recibe, nombran de forma explícita al proceso con el que se comunican:

enviar (Q, mensaje) : envía un mensaje al proceso Q
recibir (P, mensaje) : recibe un mensaje del proceso P.

Establece un enlace entre dos procesos, proporcionando seguridad en el intercambio de mensajes pero con muy poca flexibilidad.

• Los mensajes se envían y reciben a través de un entidad intermedia que recibe el nombre de buzón o puerto. Ofrece más versatilidad que la comunicación directa, ya que permite comunicación de uno a uno, de uno a muchos, de muchos a uno y de muchos a muchos. Cada buzón tiene un identificador que le distingue. Las operaciones tiene la forma:

enviar (buzón_ A, mensaje): envía un mensaje al buzón A
recibir (Buzón_ A, mensaje): recibe un mensaje del buzón A.

Modelos de sincronización

Las diferentes formas que puede adoptar la operación de *envío* del mensaje son:

- *Síncrono*: El proceso que envía sólo prosigue su tarea cuando el mensaje ha sido recibido. Necesita que ambos procesos se 'juntan' para realizar la comunicación, encuentre (rendevouz).
- *Asíncrono*: El proceso que envía sigue su ejecución sin preocuparse si el mensaje ha sido recibido. El Sistema Operativo se encarga de recoger el mensaje del emisor y almacenarlo a la espera de que lo recoja una operación de recibir.
- *Invocación remota*: Conocida como encuentro extendido, el proceso que envía sólo prosigue su ejecución cuando ha recibido respuesta del receptor.

La operación de recibir, tanto en el método de comunicación directa como indirecta, se suele realizar de modo que el proceso que ejecuta la operación toma un mensaje si este está presente y se suspende si no lo está. Para solucionar una posible espera indefinida, se proporciona una operación de recibir sin bloqueo (aceptar), de modo que si el mensaje está presente se devuelve, y si no lo está se devuelve un código de error. Una solución más adecuada es la de especificar un tiempo máximo de espera del mensaje, que una vez transcurrido, el Sistema operativo envía un código de error por agotamiento de tiempo. La operación sería:

recibir (buzón, mensaje, tiempo_ espera);

Las formas de enviar un mensaje se pueden relacionar, dos operaciones asíncronas pueden constituir una relación síncrona si se le envía una señal de reconocimiento. El procedimiento sería:

P	Q
enviar (Q, mensaje)	recibir (P, mensaje)
recibir (P, reconocimiento)	enviar (P, reconocimiento)

Almacenamiento y estructura del mensaje.

El intercambio de información se puede realizar *por valor* o *por referencia*. En la transferencia por valor se realiza una copia del mensaje desde el espacio de direcciones del emisor al espacio de direcciones del receptor, mientras que en la transferencia por referencia se pasa un puntero al mensaje. La ventaja del primero es que mantiene desacoplada la información, lo que supone mayor seguridad en la integridad. Su inconveniente son el gasto de memoria y de tiempo, que se suelen ver incrementados por el uso de una memoria intermedia. El método de sincronización de la invocación remota utiliza necesariamente la transferencia por valor, mientras que los métodos síncrono y asíncrono pueden utilizar ambos modos.

Atendiendo a la estructura de los mensajes estos se pueden considerar divididos en tres tipos, longitud fija, longitud variable y de tipo definido. El formato de longitud fija resulta en una implementación física que permite una asignación sencilla y eficaz, principalmente en las transferencias por valor, aunque dificulta la tarea de programación. Los mensajes de longitud variable son más adecuados en los sistemas donde la transferencia se realiza por punteros, ya que la longitud del mensaje puede formar parte de la propia información transmitida. La programación resulta más sencilla, a expensas de mayor dificultad en su realización. Los mensajes con definición de tipo se adaptan mejor a la comunicación con buzones. A cada buzón que utilice un proceso se le puede asignar el tipo de dato adecuado para dicho mensaje y sólo los mensajes con esta estructura pueden ser enviados por ese enlace.

La comunicación entre mensajes se puede utilizar para realizar un semáforo binario.

```

Module semáforo
type mensaje = ...; (*definición del tipo de mensaje*)
const nulo = ...; (* mensaje vacío *)

Procedure espera (var S: integer);
  var temp: mensaje;
  begin
    recibir (Sbuzón, temp);
    S := 0;
  end espera

Procedure señal (var S:integer);
  begin
    enviar (Sbuzon, nulo);
  end señal;

Procedure inicializa (var S: integer; valor: boolean);
  begin
    if valor := 1 then
      enviar (Sbuzón,nulo);
    end;
    S := valor;
  end inicializa;

begin
  crear_ buzón (Sbuzón);
end semáforo.

```

También se puede obtener una solución al problema del productor - consumidor utilizando mensajes. En esta solución los mensajes se utilizan para transmitir los datos producidos y consumidos. El sistema de mensajes es asíncrono por buzones y con una capacidad limitada de la cola:

```

process Productor_ N;
  var x : mensaje;
  begin
    loop
      produce_ datos_ y_ mensaje;
      enviar (buzón, x);
      resto_ operaciones_ P;
    end
  end Productor_ N;

process Consumidor_ M;
  var y : mensaje;
  begin
    loop
      operaciones_ consumidor;
      recibir (buzón, x);
      consume_ datos;
    end
  end Consumidor_ M;

```

La solución anterior no es válida si la cola tiene una capacidad finita pero se trata como si fuera ilimitada. En este caso un proceso que ejecuta la operación de enviar no se suspende si la cola del buzón está llena, luego la solución anterior puede llevar a que se pierdan mensajes. La siguiente solución evita esta pérdida.

```
(*Problema del Productor - Consumidor: Solución con mensajes*)
module Productor_Consumidor
type mensaje = ... ; (*Definición del tipo de datos del mensaje*)
const tamañoQ = ...; (*tamaño de la cola de los buzones*)
      nulo = ...; (*mensaje vacío*)
var n : integer;
process Productor1;
  var x: mensaje
  begin
    loop
      recibir (buzónP, x);
      produce_datos_y_mensaje ;
      enviar (buzónC, x);
      resto_operacionesP;
    end
  end Productor1;
process Consumidor1;
  var y: mensaje
  begin
    loop
      recibir (buzónC, y);
      consume_datos_y_mensaje ;
      enviar (buzónP, y);
      resto_operacionesC;
    end
  end Consumidor1;

begin
  crear_buzón (buzónP);
  crear_buzón (buzónC);
  for n : = 1 to tamañoQ do
    enviar (BuzónP, nulo);
  end
  cobegin
    Productor1;
    Consumidor1;
  coend;
end Productor_Consumidor;
```

Interbloqueo

Interbloqueo es la peor situación de error que puede ocurrir en procesos concurrentes, consiste en que dos o más procesos entren en un estado que imposibilita a cualquiera de ellos a salir del estado en el que se encuentran. A dicha situación se llega porque cada proceso adquiere algún recurso necesario para su operación a la vez que espera que se liberen otros recursos que mantienen otros procesos.

Caracterización del interbloqueo.

Para que se produzca una situación de interbloqueo se debe cumplir simultáneamente las cuatro condiciones siguientes:

1. Exclusión mutua. Los procesos utilizan de forma exclusiva los recursos que han adquirido. Si otro proceso pide el recurso debe esperar a que se libere.
2. Petición y espera. Los procesos retienen los recursos que han adquirido mientras esperan otros recursos que son retenidos por otros procesos.
3. No existencia de expropiación. Un proceso que posee un recurso no se le puede quitar.

4. Espera circular. Existe una cadena de procesos en la que cada uno tiene al menos un recurso que se solicita por el siguiente proceso de la cadena.

Prevención de interbloqueos.

Se trata de evitar que se pueda llegar a la situación de interbloqueo. Es un método muy utilizado, que puede llevar a una utilización muy pobre de los recursos. Se basa en evitar que se de una de las cuatro condiciones para que se produzca el interbloqueo (excepto la exclusión mutua que se debe mantener).

- Retención y espera: Para que no se cumpla se debe garantizar que un proceso que posee un recurso no puede pedir otro. Si necesita varios o los consigue todos o ninguno. Supone un uso pobre de los recursos y es difícil conseguir un conjunto de recursos libres.
- No existencia de expropiación: Evidentemente se permite la expropiación, si a un proceso en retención y espera tiene un recurso solicitado por otro se le puede expropiar. Sus desventaja es que puede relegar a ciertos procesos durante mucho tiempo.
- Espera circular: Para que no se produzca esta situación se asigna un número a cada recurso, por orden ascendente. Las peticiones a los recursos se realizan de forma única. Tiene la desventaja de que los recursos no se piden con el orden que se necesitan sino con uno establecido previamente.

Evitación de interbloqueos

El objetivo es el de imponer condiciones de manera que cuando se vislumbra la posibilidad de que ocurra un interbloqueo no se concedan los recursos implicado. La técnica más utilizada es la del algoritmo del banquero (Dijkstra). Este método presenta las siguientes características:

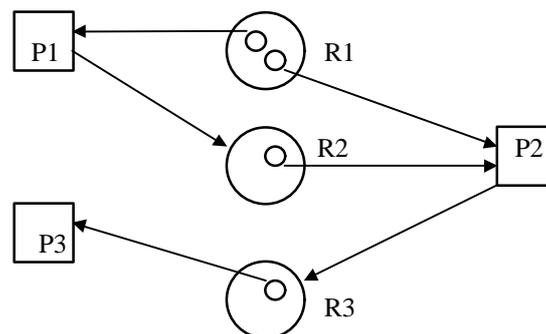
- a. Permite las condiciones de exclusión mutua, no existencia de expropiación y de retención y espera.
- b. Los procesos solicitan el uso exclusivo de los recursos que necesitan; mientras esperan a alguno se les permite mantener los recursos que disponen sin que se les puedan expropiar.
- c. Los procesos piden los recursos al sistema operativo de uno en uno.
- d. El sistema puede conceder o rechazar cada petición.
- e. Una petición que no conduce a un estado seguro es rechazada, si conduce será aceptada.

Debido a que siempre se mantiene el sistema en un estado seguro todos los procesos podrán eventualmente conseguir los recursos que necesita y finalizar su ejecución. El inconveniente es que el método es conservador y requiere conocer las necesidades máximas de los procesos.

Detección de interbloqueos.

Este método se utiliza en aquellos sistemas en los que se permite que se produzca el interbloqueo. Se procede comprobando si se ha producido el interbloqueo, si es así, se identifican los procesos y recursos afectados para poder dar la solución adecuada.

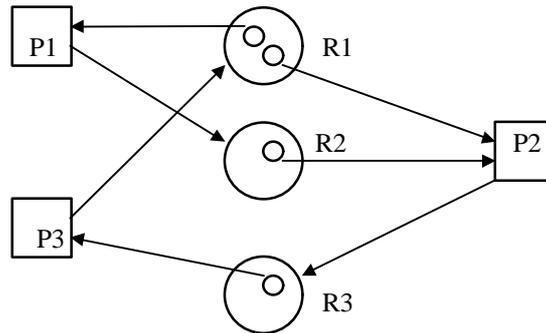
Para facilitar la detección de los interbloqueos se utiliza un grafo dirigido que indica las asignaciones de los recursos a los procesos y las peticiones que estos realizan. Los nodos del grafo son procesos y recursos; cada arco conecta un proceso con un recurso. A este grafo se le denomina *grafo de asignación de recursos del sistema*.



Los procesos se representan con cuadrados y los recursos de un mismo tipo con círculos. Dentro de los círculos hay círculos pequeños, que representan el número de recursos que hay de ese tipo. El recurso concedido se indica mediante un arco que va desde el nodo proceso solicitante hasta el recurso concedido; la propiedad de un recurso se indica mediante un arco desde el recurso al proceso.

Si el grafo no contiene ningún ciclo no hay ningún proceso que esté bloqueado, pero si lo tiene puede existir algún tipo de interbloqueo. Si sólo hay un elemento por cada tipo de recurso la existencia de un ciclo es una condición necesaria y suficiente para que haya interbloqueo. Si cada tipo de recurso tiene varios elementos, entonces la condición de que haya un

ciclo es necesaria, la condición de suficiente es la existencia de un ciclo en el que no hay ningún camino que salga de alguno de los nodos que lo forman que a su vez no sea un ciclo.



El grafo contiene dos ciclos:

[R1, P1, R2, P2, R3, P3, R1]

[R1, P2, R3, P3, R1]

Se puede utilizar un método de reducción del grafo, con el fin de detectar el interbloqueo y de obtener los procesos que pueden completarse y los que permanezcan bloqueados.

Recuperación de interbloqueos.

Una vez detectado el interbloqueo se debe romper para que los procesos puedan finalizar su ejecución y liberar así los recursos. Para la ruptura de la espera circular hay varias opciones. La solución de suspender algún recurso bloqueado tomar sus recursos y reanudarlo cuando se deshaga el interbloqueo sólo resulta factible en casos muy particulares; por ello las dos opciones que se suelen utilizar son reiniciar uno o más procesos bloqueados u expropiar los recursos de algunos de los procesos bloqueados de forma sucesiva hasta que se salga del interbloqueo.

4. GESTIÓN DE LA MEMORIA

La misión del gestor de memoria es la asignación de memoria principal a los procesos que la soliciten. El espacio vacante se puede utilizar para cargar procesos que ya están preparados para su ejecución, de forma que el planificador está en mejores condiciones de preparar las tareas que se van a ejecutar. La utilización de los recursos del sistema, y en general, el rendimiento del sistema, va a estar condicionado por la eficacia en la gestión de la memoria.

Sistemas elementales de gestión de la memoria

El esquema de gestión de la memoria más simple que se puede considerar es aquel que no tiene ningún gestor. En este caso el usuario se encuentra sólo con la máquina y tiene un control completo sobre el espacio total de memoria. Tiene como ventajas máxima flexibilidad al usuario con control total sobre el uso de la memoria. Máxima simplicidad y coste mínimo. No es necesario disponer de un computador específico; ni tan siquiera de sistema operativo. Sus limitaciones son que no da ningún servicio, que el SO no tiene control sobre las interrupciones y que no hay un monitor residente para procesar las llamadas al sistema o para el tratamiento de errores. Este sistema fue usual hasta el comienzo de la década de los 60, pero en la actualidad no se utiliza salvo en algunos sistemas dedicados en los que el usuario requiere flexibilidad y simplicidad, y programa él mismo sus propias rutinas de soporte.

El siguiente esquema en simplicidad se obtiene al dividir la memoria en dos secciones una para el proceso del usuario, y otra para la parte del sistema operativo que debe de estar residente en memoria (monitor). Este esquema fue bastante común, con alguna variante, en los sistemas operativos de los microcomputadores de proceso único, como las primeras versiones de MS-DOS para el PC de IBM o su antecesor el CP/M. En el MS-DOS para el PC de IBM, la memoria se encuentra dividida en tres zonas la parte inferior de la RAM está dedicada al sistema operativo, la parte central, dedicada al único programa de usuario, la parte superior, en ROM, para los controladores de dispositivos. El programa en ROM se llama BIOS (*Basic Input Output System*), Sistema Básico de E/S. Este esquema supone que sólo se ejecuta un proceso cada vez. Las grandes estaciones de trabajo y grandes computadores y últimamente los computadores personales se ha impuesto la multiprogramación las razones que la hacen conveniente son:

1. Facilitar la programación de una aplicación al dividirla en varios procesos.
2. Poder dar servicio interactivo a varios usuarios al mismo tiempo o el optimizar el tiempo del procesador cuando un proceso está a la espera de una operación de entrada/salida.

Gestión de la memoria con particiones fijas

Una forma de realizar la multiprogramación consiste en dividir la memoria física disponible en varias particiones. El tamaño de las particiones se puede fijar con anterioridad a la ejecución de los programas o efectuarse dinámicamente en fase de ejecución. La asignación de memoria con particiones fijas se denominó MFT, (*Multiprogramming with a Fixed number of Tasks*) Multiprogramación con un número fijo de tareas, y fue utilizada durante muchos años en los grandes computadores de IBM con OS/360, por lo que también se la conoce como OS/MFT. La división de la memoria se realiza antes de la ejecución de los programas cuando se inicia la sesión. El número y tamaño de las particiones se determina teniendo en cuenta los factores siguientes:

1. Capacidad de la memoria física disponible y el *grado de multiprogramación* deseado (número máximo de procesos activos en el sistema).
2. Tamaños típicos de los procesos ejecutados más frecuentemente.

Principio de operación

La forma de operación es la siguiente:

- a. Cuando llega una tarea, ésta se pone en la cola de tareas.
- b. El planificador de tareas tiene en cuenta los requerimientos de memoria de cada una de ellas y las particiones de memoria disponibles.
- c. Si una tarea tiene espacio disponible en memoria, se ubica (o reubica) en una partición, y puede competir por el uso de la UCP.
- d. Cuando se termina la tarea, se libera la partición de memoria que ocupa.

Las organizaciones posibles para la asignación de memoria a las tareas son: clasificarlas todas según sus requerimientos de espacio. De esta forma, cada partición de memoria tendrá una cola de tareas y las tareas se incluyen en la cola de la partición de memoria correspondiente a sus exigencias de memoria. Mantenerlas todas las tareas en una sola cola. Cuando se libera una partición se puede asignar la partición a la primera tarea que quepa. El principal inconveniente se produce cuando se asigna una gran partición a una tarea con pocos requerimientos de memoria. Otra solución es asignar la partición a la tarea más grande que quepa. En este caso se discrimina a las tareas pequeñas, cuando lo recomendable es que se les dé el mejor servicio y no el peor. Como solución se puede tener una partición de tamaño reducido para la ejecución de las tareas pequeñas. O fijar como regla que una tarea no se excluya más de un número de veces fijado a priori por el sistema operativo.

Intercambio

El *intercambio* (*swapping*) hace referencia a las operaciones de eliminar de la memoria principal procesos suspendidos, llevarlos al disco y cargar del disco a la memoria principal procesos para su ejecución. Las responsabilidades del intercambiador son:

- Selección de los procesos que hay que eliminar de la memoria principal. Se consideran cuestiones como:
 - Tamaño.* La selección se hace entre aquellos que ocupan particiones suficientemente grandes para los procesos que se les debe asignar espacio en memoria.
 - Prioridad.* Son más aptos para intercambiar aquellos con baja prioridad.
 - Estado de espera.* Procesos que esperan sucesos lentos tienen una mayor probabilidad de ser suspendidos durante un tiempo suficientemente grande.
 - Tiempo de permanencia en memoria* y si se ha ejecutado mientras ha estado en memoria. Si no hay peligro de estar eliminando procesos casi inmediatamente después de ser cargados.
- Selección de los procesos que hay que cargar en la memoria principal. Aspectos:
 - Tiempo que ha permanecido en memoria secundaria.
 - Prioridad.
 - Satisfacción del tiempo mínimo de residencia en disco desde la descarga.
- Asignación y gestión del espacio de intercambio.

Archivo de intercambio

Un proceso se prepara para su ejecución y se envía al SO en la forma de un archivo que contiene un programa en versión ejecutable y sus correspondientes datos. Este archivo puede también contener atributos del proceso, como prioridad y requerimientos de memoria. Este archivo es llamado algunas veces imagen del proceso. Como un proceso generalmente modifica su pila y sus datos cuando se ejecuta, un proceso ejecutado parcialmente tiene una imagen en tiempo de ejecución diferente de la inicial almacenada en el disco.

La parte modificable del estado del proceso consiste en los contenidos de sus datos y posiciones de la pila, así como de los registros del procesador. Como la imagen del proceso estático se utiliza para la activación inicial, la imagen debe guardarse en un lugar diferente para no sobrescribir el proceso original. Por este motivo se debe disponer de un archivo de intercambio separado para almacenar la imagen dinámica del proceso. Dos opciones básicas en el diseño del archivo de intercambio son:

- Un único archivo de intercambio para el sistema completo.
 - Se suele crear al inicializar el sistema.
 - Se suele situar en un disco rápido, para reducir los tiempos de intercambio.
 - La elección del tamaño del archivo es muy importante. Una solución está en crear un archivo no muy grande: se dispone de más espacio en el disco para programas. El problema es que si no hay espacio suficiente, puede que un proceso no se active y cause problemas.
- Diferentes archivos de intercambio, uno para cada proceso.
 - Se suelen crear dinámicamente en el tiempo de generación del proceso o estáticamente en el tiempo de preparación del programa.
 - Ventajas: se elimina el problema de dimensionamiento del archivo de intercambio y los errores de sobrepasar la capacidad del archivo en tiempo de ejecución, además de no imponer restricciones sobre el número de procesos activos.
 - Desventajas: necesidad de más espacio de disco, un acceso más lento y un direccionamiento más complicado de los archivos de intercambio (están dispersos en la memoria).

En cada caso, el espacio de intercambio para cada proceso susceptible de poderse transferir está generalmente reservado, y asignado estáticamente en el momento de creación del proceso, para evitar un incremento adicional en el tiempo de intercambio.

Reubicación y protección

El término *reubicable* se refiere a la posibilidad de cargar y ejecutar un programa dado en un lugar arbitrario de memoria. Se debe distinguir entre *direcciones lógicas o virtuales*, aquellas percibidas por el programador, son relativas; y *direcciones físicas*: en las que el programa y sus datos se almacenan durante la ejecución.

La solución al problema de la reubicación pasa por considerar dos tipos de reubicaciones la *estática* que se realiza antes o durante la carga del programa en memoria, mediante un enlazador o cargador reubicable, y la reubicación *dinámica* que implica que la asignación del espacio de direcciones virtuales al espacio de direcciones físicas se realiza en tiempo de ejecución, generalmente con ayuda de dispositivos físicos específicos destinados a este fin. Las direcciones se recalculan mediante *registros base*. El registro base se carga con la dirección de inicio de la partición imagen del proceso. La dirección

física será la dirección lógica dada por el proceso más el valor del registro base. La gran ventaja es que el programa se puede trasladar de partición en tiempo de ejecución, y sólo hará falta variar el registro base.

Queda por resolver el problema de la *protección*, para evitar que un programa de forma mal intencionada puede construir una dirección física que se salga fuera de la partición que se le ha asignado. Hay tres soluciones:

1. Uso de un Registro límite: Se usa para la reubicación dinámica. Este registro contiene el límite de direcciones lógicas admisibles por el programa. Antes de sumar a la dirección lógica el registro base, se comprueba que la dirección lógica esté permitida. Si no lo está se envía una señal al sistema indicando que se ha producido un error de direccionamiento.
2. Uso de Registros Bordes: Se usa, principalmente, para la reubicación estática. Se consideran registros bordes para cada partición, los cuales se cargan con las direcciones físicas de comienzo y final de la partición.
3. Otro método diferente consiste en comprobar el acceso en la memoria misma. La memoria se divide en bloques y a cada bloque se le asigna un código o contraseña. Cuando un programa intenta acceder a un bloque su código de acceso debe coincidir con el del bloque de memoria.

Selección del tamaño de la partición; fragmentación de la memoria

La decisión inicial generalmente se basa en una suposición razonable de los requerimientos de los programas de entrada. El problema principal con las particiones fijas, es llegar a un buen compromiso entre los tamaños de las particiones y la memoria requerida para los trabajos. Existen dos tipos de desaprovechamiento de la memoria:

Fragmentación Interna. Es aquella parte de la memoria que no se está usando, pero que es interna a una partición y no puede ser aprovechada.

Fragmentación Externa. Es aquella partición de la memoria que no se está usando, porque es demasiado pequeña para las tareas que están esperando a ser ejecutadas.

La selección de los tamaños de las particiones es un compromiso entre estos dos tipos de fragmentación.

GESTIÓN DE MEMORIA CON PARTICIONES VARIABLES

Las particiones fijas tienen el inconveniente de que hay que determinar sus mejores tamaños con el fin de minimizar la fragmentación interna y externa. La solución de este problema está en permitir que los tamaños de las particiones varíen dinámicamente. Este método de gestión de memoria se conoce como asignación de la memoria con particiones variables.

Principio de operación

El SO mantiene una tabla que indica que partes de la memoria están ocupadas y cuales están libres. Inicialmente toda la memoria está disponible y se considera como un gran bloque. Cada vez que llega una tarea se busca un bloque lo suficientemente grande y se le asigna a la tarea la cantidad necesaria de memoria, quedando el resto del bloque como libre para futuras demandas. El funcionamiento es:

1. Cuando un proceso llega y necesita memoria, se busca en este conjunto un bloque de memoria libre que sea suficientemente grande para dicho proceso. Si el bloque elegido es más grande, se parte en dos: uno se asigna al proceso y el otro se devuelve al conjunto de bloques libres.
2. Cuando un proceso termina, libera su partición de memoria que pasa a formar parte del conjunto de bloques libres. Si ese nuevo bloque libre es adyacente a otro u otros bloques, se unen todos ellos en uno solo más grande. En este caso se comprueba si hay procesos esperando memoria y si este nuevo espacio libre recombinado satisface los requerimientos de memoria de cualquiera de estos procesos que esperan.

Si se pueden mover todos los bloques libres hacia abajo (o hacia arriba) en la memoria, siempre es posible obtener un solo gran bloque. Este proceso es la llamada *compactación de la memoria*. Pero generalmente no se realiza pues consume mucho tiempo de UCP

Un problema importante aparece cuando un proceso utiliza asignación dinámica y por lo tanto puede crecer en tiempo de ejecución. Entonces se le debe asignar espacio libre contiguo para que pueda crecer. Incluso puede ser necesario intercambiar algunos procesos para que este espacio libre esté disponible.

Sistemas de registro del uso de la memoria

Nos interesa usar un sistema que sea eficiente, tanto en tiempo para la asignación como en el aprovechamiento de la memoria. Hay varios sistemas a considerar:

Mapas de bits: La memoria se divide en unidades de asignación (de varios bytes hasta Kbytes) y a cada unidad de asignación se le asocia un bit en el mapa de bits. Un "0" significa que esa unidad está libre; un "1" que está ocupada. Tienen el inconveniente que para buscar una partición suficientemente grande para un proceso se debe recorrer el mapa de bits buscando una cadena de ceros que represente esa partición suficientemente grande.

Listas enlazadas se mantiene una lista enlazada de registros, donde cada registro representa un bloque libre u ocupado de memoria y tiene como campos la dirección de inicio, el tamaño y una referencia a si el bloque correspondiente está libre u ocupado. La lista enlazada suele estar ordenada por direcciones. Entre las estrategias más comunes de asignación de memoria están las siguientes:

Primero en ajustarse: El gestor de memoria asigna al proceso entrante el primer bloque de memoria libre suficientemente grande que encuentra al recorrer la lista. La parte sobrante del bloque se devuelve como bloque libre.

Próximo en ajustarse: Funciona de la misma forma que el anterior, pero manteniendo un registro del lugar donde se encuentra una partición adecuada, asignando no la primera sino la siguiente que se ajusta (si existe).

Mejor en ajustarse: Busca en toda la lista y asigna el bloque suficientemente grande más pequeño que encuentra. Produce restos más pequeños pero es más lento que los anteriores. También produce un mayor desperdicio de memoria que los otros dos algoritmos, puesto que tiende a rellenar la memoria con particiones libre pequeñas sin utilidad alguna.

Peor en ajustarse: Busca en toda la lista y asigna el bloque mayor entre los que existen. Produce los residuos más grandes. Por simulación se ha demostrado que no tiene un rendimiento superior a los otros algoritmos.

Estos algoritmos se pueden agilizar si se contemplan algunas variantes de los mismos. Por ejemplo, el mantenimiento de dos listas independientes: una para los procesos y otra con los bloques libres. Se aumenta la velocidad de asignación de memoria, pero tiene una mayor complejidad y presenta una disminución de la velocidad al liberar memoria (se debe eliminar de la lista de procesos e insertarse en la de bloques libres).

Otro método de tener un registro del uso de la memoria es mediante el *sistema de los asociados*, que aprovecha el hecho de trabajar con direcciones binarias para hacer más rápida la fusión entre particiones libres adyacentes. Se mantiene una lista de las particiones libres, cuyos tamaños son siempre potencias de dos hasta el total de la memoria. Cuando un proceso solicita la entrada se le asigna el bloque de memoria potencia de dos menor pero suficientemente grande. Si es necesario se rompe en dos partes iguales un bloque de tamaño superior. Cuando se libera memoria se unen los bloques sólo cuando están asociados, es decir, provienen de la misma partición de bloque, son del mismo tamaño y formarán uno de tamaño inmediatamente superior.

La principal ventaja del sistema de los asociados es su mayor rapidez para decidir la fusión de dos bloques, su desventaja es que no sólo produce fragmentación externa, sino también fragmentación interna.

Análisis de la gestión de la memoria con particiones variables

La fragmentación se produce por la diferencia de vida de los procesos, que implica que el patrón para devolver los bloques libres es distinto del orden de asignación. Después de algún tiempo en operación, los sistemas con asignación dinámica de memoria tienden a un estado de equilibrio en el que la memoria desperdiciada por un determinado algoritmo de asignación se puede estimar y ser usada como un parámetro de comparación. Así para el algoritmo “primero en ajustarse” con fusión de los bloques libres adyacentes, Knuth demostró que si el promedio de procesos en memoria es n , el promedio de particiones libres es $n/2$. Este resultado se conoce como *regla del cincuenta por ciento*. Otra medida útil es la *fracción de memoria no utilizada*. Para su cálculo se definen los parámetros siguientes:

- f , la fracción de memoria ocupada por las particiones libres.
- s , el tamaño medio de los n procesos.
- $k \cdot s$, el tamaño medio de las particiones libres para algún valor de $k > 0$.

Si el tamaño total de la memoria es de m bytes, las $n/2$ particiones libres ocuparán $m - n \cdot s$ bytes. Es decir:

$$\frac{n}{2} \cdot k \cdot s = m - n \cdot s$$

Luego:

$$m = n \cdot s \cdot \left(1 + \frac{k}{2}\right)$$

La fracción de memoria libre es el número de particiones libres, $n/2$, por el tamaño medio de las particiones libres, $k \cdot s$, dividido por el tamaño de toda la memoria, m :

$$f = \frac{\frac{n \cdot k \cdot s}{2}}{m} = \frac{\frac{n \cdot k \cdot s}{2}}{n \cdot s \cdot \left(1 + \frac{k}{2}\right)} = \frac{k}{k + 2}$$

Esta fórmula demuestra que mientras el tamaño medio de las particiones libres sea una fracción apreciable del tamaño promedio de los procesos, se desaprovechará una cantidad importante de memoria.

PAGINACIÓN

El problema principal de los anteriores sistemas de gestión de la memoria es la fragmentación externa, esta situación se presenta cuando la memoria disponible no es contigua, sino que está fragmentada en muchos bloques repartidos a lo largo del espacio de direcciones. Dado que la memoria asignada a un proceso debe ser contigua, esta memoria que está libre pero dispersa no se puede usar. Este problema tiene dos soluciones generales:

- Compactación de memoria, que no suele usarse por su coste.
- Paginación, que permite que la memoria de un programa no sea contigua, de forma que siempre que se disponga de espacio, aunque éste no sea adyacente, se pueda asignar a un programa.

Principio de operación

La memoria se divide conceptualmente en un número de bloques de tamaño fijo, denominados *marcos de página*. El espacio de direcciones virtuales de un proceso también se divide en bloques de tamaño fijo, llamados *páginas*, del mismo tamaño que los marcos de página. La asignación de memoria consiste en encontrar un número suficiente de marcos de página sin usar para cargar las páginas de los procesos solicitados. Mediante un mecanismo de traducción de direcciones, se asignan las páginas virtuales a su contrapartida física. Como cada página se asigna por separado, no es necesario que los marcos de página asignados a un único proceso estén situados en zonas contiguas de la memoria física. Se consideran marcos de páginas y páginas que sean potencias de 2. Por ejemplo, la dirección lógica 0020FF se descompone en dos partes, la primera (002) indica la página donde buscar la dirección física del marco de página correspondiente; la segunda (0FF) el desplazamiento relativo dentro de la página. Cuando se busca la página 002 en la tabla de páginas se obtiene el marco de página 104 al cual se añade sin modificación el desplazamiento 0FF. El sistema operativo mantiene el estado de cada marco de página de la memoria mediante un mapa de la memoria física estructurado como una tabla estática. Esta tabla tiene un número fijo de entradas, que es igual al número de marcos de página (que viene dado por la relación entre la capacidad de la memoria física instalada y el tamaño de cada página). Para cada entrada se indica si el marco de página correspondiente está asignado o libre. En este sistema no se tiene fragmentación externa. Sin embargo, sí se tiene algo de fragmentación interna, lógicamente. Cuanto más pequeña es la página, menor es la fragmentación interna, pero un tamaño demasiado pequeño puede llevar a una tabla de páginas excesivamente grande.

Implementación de las tablas de páginas

El objetivo de las tablas de páginas es asociar a las páginas un marco de página. Esto se puede ver como una función matemática, que tiene como argumento el número de página y que da como valor el número del marco de página correspondiente. Con el resultado de esta función para construir la dirección física se reemplaza el campo de la página en la dirección virtual por el marco de página. Se plantean dos cuestiones importantes:

1. El tamaño de la tabla de páginas, que puede ser demasiado grande. Es consecuencia directa de la capacidad de los computadores actuales que pueden realizar direccionamientos de 32 o 64 bits. Por ejemplo, con páginas de 4 Kb y un espacio de direcciones de 32 bits se tendrían 1 millón de páginas.
2. El tiempo de asignación, que debe ser rápido. Para cada referencia a memoria, se debe realizar la asociación entre dirección física y dirección lógica, por lo tanto se hace necesario que este proceso sea lo suficientemente rápido para que no suponga un cuello de botella en el computador.

Conceptualmente, la realización más sencilla es tener una sola tabla de páginas estructurada como un conjunto de registros dedicados la ventaja es que sólo se realiza una referencia a la memoria durante la asociación, su desventaja es el coste de la circuitería extra cuando las tablas de páginas son grandes. Además, la carga de la tabla de páginas en cada cambio de contexto (por ejemplo, por la reubicación de un proceso en memoria) puede hacer disminuir el rendimiento. Este método sólo es adecuado cuando la tabla de páginas es pequeña.

La utilización de memoria asociativa se basa en el hecho de que la mayoría de los programas tienden a hacer un gran número de referencias a un número pequeño de páginas y no al revés. La tabla de páginas se encuentra cargada en memoria y un registro apunta al principio de la tabla. La memoria asociativa es un dispositivo hardware que consta de un número pequeño de entradas (no suelen ser más de 32). Cada entrada tiene la siguiente información relativa a una página:

1. Número de página.
2. Marco físico donde se localiza la página.
3. Código de protección: sólo lectura, lectura/escritura, etc.
4. Bit de modificación.

Cuando se presenta una dirección lógica se consultan todas las entradas de la memoria asociativa en paralelo. Dos posibilidades, si el número de la página se halla en la memoria asociativa y el acceso no viola los bits de protección, el marco de página se toma directamente de la memoria asociativa. Si se viola la protección o el número de página no se encuentra en la memoria asociativa se recurre a la tabla de páginas para obtener el marco de página. Además la referencia a la página puede sustituir a otra en la memoria asociativa, de forma que si se vuelve a utilizar esta página, su información ya se encuentre en la memoria asociativa.

Se calcula que el porcentaje de veces que un número de página se encuentra en la memoria asociativa (*razón de encuentros*) se encuentra relacionado con el número de registros. Si el sistema es multiprogramado cada proceso debe tener su propia tabla de páginas y es necesario dar mecanismos para que al ejecutarse un nuevo proceso no utilice los marcos de página de otro proceso. Se suelen adoptar dos soluciones:

- a. Proporcionar una instrucción máquina que invalide la memoria asociativa, es decir, que borre todos los bits de validación.
- b. Realizar la memoria asociativa con un campo de identificación del proceso y añadir un nuevo registro con el identificador del proceso activo.

Compartición de páginas y protección

Una de las ventajas de la paginación es la posibilidad de compartir código, lo cual es interesante sobre todo en entornos de tiempo compartido. Para ser compatible el código debe ser *reentrante*, es decir, no *automodificable*, de forma que no cambia nunca durante la ejecución. De esta forma, dos o más procesos pueden ejecutar el mismo código a la vez, manteniendo copia de sus registros y de sus datos, ya que estos varían de un proceso a otro.

En un sistema con paginación, la protección de memoria se realiza mediante bits de protección asociados a cada una de las páginas. Estos bits se encuentran en la tabla de páginas y definen la página como de lectura, escritura o de sólo lectura, de forma que al acceder a la página, mientras se calcula la dirección física, se verifica el valor de este bit para no permitir escribir en una página de sólo lectura.

Segmentación

La segmentación es un sistema de gestión de la memoria donde el espacio de direcciones lógicas es un conjunto de segmentos, con diferentes nombres y tamaños. Las direcciones especifican el nombre del segmento y el desplazamiento dentro del segmento; por ello el usuario debe especificar cada dirección mediante dos cantidades, el nombre del segmento y un desplazamiento. Un segmento es un objeto lógico, conocido por el programador y que lo utiliza como tal. Un segmento tiene una determinada estructura de datos (pila, cola, etc.) o un procedimiento o módulo, pero normalmente no contiene una mezcla de varios.

Principio de operación

El compilador construye los segmentos automáticamente de acuerdo a la estructura del programa. El cargador tomará todos los segmentos y les asignará números de segmento. Por propósito de reubicación cada uno de los segmentos se compila comenzando por su propia dirección lógica 0. Las direcciones lógicas constarán de un número de segmento y un desplazamiento dentro de él. En este contexto, el espacio de direcciones es bidimensional, pero la memoria física mantiene su organización lineal unidimensional, luego es necesario disponer de un mecanismo que convierta las direcciones lógicas bidimensionales en su dirección física equivalente unidimensional. Esta asignación se efectúa mediante una *tabla de segmentos*. En la tabla se mantienen registrados los descriptores de cada segmento: la base, correspondiente a la dirección física donde se inicia el segmento (obtenida durante la creación de la partición) y el tamaño del segmento que se usa para saber si el desplazamiento es válido. La dirección física se obtiene añadiendo (sumando) el desplazamiento a la base del segmento. Este desplazamiento no debe sobrepasar el tamaño máximo del segmento. Si lo superará se produciría un error de direccionamiento.

Una tabla de segmentos que se mantiene en registros especiales puede ser referenciada muy rápidamente: la suma a la base y la comparación con el tamaño se puede hacer simultáneamente. Si el número de segmentos es grande la tabla de segmentos se deja en memoria y un *registro base de la tabla de segmentos* apunta a la tabla. Además, como el número de segmentos usados por un programa puede variar, también se emplea un *registro de la longitud de la tabla de segmentos*, que contiene el número de segmentos que hay en la tabla. En este caso, ante una dirección lógica, lo primero que se hace es comprobar que el número de segmento es válido (menor que la longitud de la tabla de segmentos). Se suma el número de segmento al valor del registro base de la tabla de segmentos para obtener la dirección de memoria para la entrada de la tabla de segmentos. Esta entrada se lee de memoria y se comprueba que el desplazamiento no rebasa la longitud del segmento y se calcula la dirección física como la suma del registro base de la tabla de segmentos y del desplazamientos, lo que requiere dos referencias a memoria (al igual que la paginación), provocando una disminución de la velocidad del computador.

Al igual que en la paginación se puede utilizar una memoria asociativa para mejorar el rendimiento. La segmentación no produce fragmentación interna, pero si externa, la cual ocurre cuando todos los bloques de memoria libre son muy pequeños. Para acomodar un segmento

Protección y compartición

Una ventaja de la segmentación es la posibilidad de asociar protección a los segmentos. Puesto que los segmentos son porciones diferentes del programa definidas semánticamente, todas las entradas al segmento se usan de la misma manera. De esta forma, los segmentos de código se pueden definir de sólo lectura, mientras que un segmento de datos se puede definir de lectura escritura. La segmentación también facilita la compartición de código o datos entre varios procesos (por ejemplo, las librerías).

Los segmentos son compartidos cuando la entrada a la tabla de segmentos de dos procesos apuntan a la misma posición física. De esta forma, cualquier información se puede compartir si se define en un segmento.

Sistemas combinados de paginación – segmentación

Principales características de los esquemas de paginación y de segmentación:

- a. El programador no necesita saber que se está utilizando la paginación; pero si dispone de segmentación sí necesita saberlo.
- b. La paginación sigue un esquema lineal, con un único espacio de direcciones, la segmentación proporciona un espacio bidimensional con muchos espacios de direcciones.
- c. En paginación no se distingue el código de los datos. En cambio, en la segmentación sí se distinguen, pudiéndose proteger de forma independiente.
- d. La segmentación facilita la compartición de código y datos, así como el uso de estructuras de datos de tamaños fluctuantes.
- e. En ambas el espacio de direcciones puede exceder de la memoria física (Memoria virtual) .

Existen sistemas que combinan los dos esquemas para coger lo mejor de ambos. Una técnica muy extendida consiste en usar la segmentación desde el punto de vista del usuario pero dividiendo cada segmento en páginas de tamaño fijo para su ubicación. Esta técnica es muy útil cuando los segmentos son tan grandes que resulta muy difícil mantenerlos completos en memoria. Con esta técnica se elimina la fragmentación externa y se hace más fácil el problema de la ubicación ya que cualquier marco desocupado se puede asignar a una página. La principal diferencia con la segmentación pura es que la tabla de segmentos no contiene la dirección base del segmento, sino la dirección base de una tabla de páginas para ese segmento. Cada segmento tiene su propia tabla de páginas. Sin embargo, cada segmento tiene una longitud, dada en la tabla de segmentos, de forma que el tamaño de la tabla de páginas no tiene que ser el máximo, sino un tamaño en función de la longitud del segmento. De esta forma, en promedio, se tiene una fragmentación interna de media página por segmento. Se elimina la fragmentación externa, introduciendo fragmentación interna y aumentando el espacio de tablas.

Memoria virtual

Proviene de la idea de permitir la ejecución de procesos que puedan no estar cargados completamente en memoria. La memoria virtual consiste en un sistema de gestión de la memoria que permite que el espacio de direcciones virtuales de un proceso activo sea mayor que la memoria física disponible. El SO mantiene en la memoria principal aquella parte del proceso que se utiliza en cada momento, el resto permanece en el disco.

Principio de operación

Uno de los sistemas más comunes de memoria virtual es el denominado *demanda de página*, que es similar a un sistema de paginación con intercambio. Los procesos se inician sin ninguna página en memoria. Cuando se intenta ejecutar la primera instrucción se produce un *fallo de página* que provocará que el sistema operativo tenga que cargar la página desde el disco. Después de un cierto tiempo el proceso tiene la mayoría de las páginas que necesita y la ejecución tiende a desarrollarse con un número relativamente pequeño de fallos de página. Cuando se produce un fallo de página el procedimiento a seguir es el siguiente:

1. Se verifica si la dirección es válida. Si fuera inválida se enviaría una señal al proceso o se abortaría.
2. Si es una referencia válida, pero aún no se ha traído esa página, el SO detecta un fallo de página y determina la página virtual requerida para traerla. En este caso la instrucción queda interrumpida y se guarda el estado del proceso, para poder continuarlo en el mismo lugar y estado.
3. Se selecciona un marco libre, si no existe ninguno, se ejecuta un algoritmo de sustitución de página.
4. Si el marco de página está ocupado, el planificador transfiere la página al disco y se produce un cambio de contexto; se suspende el proceso causante del fallo y se permite la ejecución de otro, hasta que termine la transferencia al disco. El marco se marca como ocupado, evitando que se use con otros fines.
5. Cuando el marco queda limpio (ya sea inmediatamente o después de ser escrito en disco) el SO examina la dirección en el disco donde se encuentra la página necesaria y planifica una operación de lectura de la misma. Mientras, el proceso continúa suspendido.
6. Cuando se completa la lectura del disco, la tabla de páginas se actualiza para indicar que ya se dispone de la página en la memoria.
7. La instrucción que produjo el fallo regresa al estado de comienzo y se planifica su ejecución, pudiendo accederse a la página como si siempre hubiera estado en la memoria.

Se tienen que imponer algunas restricciones sobre la arquitectura, ya que es esencial poder continuar cualquier instrucción después de un fallo de página. Si se produce al buscar la instrucción, no suele ser problemático, pues resuelto el fallo de página se puede volver a buscar la instrucción como si no hubiese habido fallo. Si se produce cuando se está buscando un operando de una instrucción, para continuar el proceso se tiene que traer otra vez la instrucción, decodificarla y, a continuación, volver a buscar y traer el operando. Este proceso no es permitido por todas las arquitecturas.

Conceptos de memoria virtual

Una gestión de la memoria basada en el mecanismo de demanda de página tiene un considerable efecto sobre el rendimiento del computador. El tiempo promedio de acceso (tpa) es

$$tpa = (1 - p) \times am + p \times fp$$

donde am es el tiempo de acceso a memoria, p es la probabilidad de que ocurra un fallo de página y fp es el tiempo que lleva resolver un fallo de página. Para que el computador no sufra una gran degradación en su rendimiento el porcentaje de fallos de páginas se debe mantener a un nivel muy bajo. La realización de la memoria virtual mediante paginación supone mantener una tabla de páginas para cada uno de los procesos activos. La asignación de sólo un subconjunto de marcos de páginas reales al espacio de direcciones virtuales requiere disponer de estrategias apropiadas en la gestión de la memoria virtual. Por ello, se deben tener políticas de:

- a. Asignación de memoria real a cada uno de los procesos activos.
- b. Búsqueda de lectura del disco de los correspondientes elementos.
- c. Sustitución de elementos en memoria principal por los nuevos elementos que se tren del disco.
- d. Ubicación de los nuevos elementos.

Políticas de sustitución de páginas

Al producirse un fallo de página, el sistema operativo debe elegir una página de las que están en memoria, para sustituirla por la que se solicita. Si la página se retira ha sido modificada en la memoria se escribe en el disco para mantener actualizada la copia, si no ha sido modificada, no hay que volverla a escribir. La página que entra en memoria se escribe directamente encima de la que se retira.

Algoritmo de sustitución FIFO: primero en entrar, primero en salir

La página que más tiempo lleva en memoria es la que se sustituye por la página que se debe cargar. Se utiliza una cola para saber que página es la que se debe sustituir. Cada nueva página se coloca el inicio de esa cola y se saca la que está al final. Este algoritmo no siempre funciona correctamente. Además, sufre de lo que se denomina *anomalía de Belady* que consiste en aumentar los fallos de páginas al aumentar el número de marcos de páginas para asignación. Hay una clase de algoritmos que no presentan nunca esta anomalía y se denominan *algoritmos de pila*: se puede demostrar que el conjunto de páginas en memoria para n marcos es un subconjunto del conjunto de páginas que se tendrían con $n+1$ marcos.

Algoritmo de sustitución óptimo

Tiene que producir menos fallos de página que cualquier otro algoritmo posible. Además, no debe presentar la anomalía de Belady. Se puede enunciar como: sustituir aquella página que tardará más en volverse a utilizar. El SO no sabe a priori cual será la página que se usará más tarde. Por lo tanto, no es posible llevarlo a cabo.

Algoritmo de sustitución LRU: la menos usada últimamente

Una forma de aproximar el algoritmo óptimo consiste en estudiar el futuro a partir del pasado. Es decir, considerar que las páginas usadas más frecuentemente en las últimas instrucciones se usarán próximamente. Es realizable en teoría pero no es barato ya que necesita mucho apoyo de recursos de la máquina. El problema es mantener un orden según el tiempo que hace que se usó la página. Hay dos posibles realizaciones:

1. Mediante una pila que mantiene los números de las páginas. En el fondo de la pila se halla la página a eliminar. Se puede implementar mediante una lista doblemente enlazada. Supone mucho gasto para actualizar la pila tras eliminar una página.
2. Usando registros contadores. Cada entrada de la tabla de páginas tiene asociado un registro con el en tiempo en el que se usó por última vez esa página. Se sustituye la página cuyo contador marca un tiempo menor (más antiguo).

Algoritmo de sustitución de la segunda oportunidad

Es una modificación del algoritmo FIFO. Se hace uso de un bit de referencia. Se examina el bit de referencia de la última página en la cola. Si el bit de referencia es 0, la página no se ha utilizado además de ser antigua, y se sustituye. Si el bit de referencia es 1, entonces se pone a cero y la página se coloca al final de la cola: no se sustituye y se pasa a comprobar la siguiente página. Si todas las páginas tienen el bit de referencia a 1, se vuelve a comprobar el final de la cola (cola circular). Degenere en un algoritmo FIFO puro si todas las páginas tienen alguna referencia.

Otros algoritmos de sustitución

La mayoría son aproximaciones del LRU, además del bit de referencia, se puede considerar un *bit de modificación*, que se activa cuando se escribe una palabra en la página, ante la búsqueda de la página a sustituir, hay cuatro posibilidades:

1. página no referenciada, ni modificada (00=0).
2. página no referenciada, pero sí modificada (01=1).

3. página referenciada, pero no modificada ($10=2$).
4. página referenciada y modificada ($11=3$).

Se sustituye la página con “número” menor. En caso de empate se sigue el algoritmo FIFO. Otra posibilidad es construir el algoritmo LRU mediante un programa. Este es el algoritmo denominado *de uso no frecuente (NFU)*. Necesita una variable tipo contador asociada a cada página, con valor inicial 0. Periódicamente el SO suma el bit de referencia al contador. Cuando ocurre un fallo de página se elige la página con el valor mínimo en el contador. El principal problema de este algoritmo es que no olvida. Así páginas usadas frecuentemente hace tiempo mantienen un valor alto en el contador y no son sustituidas a pesar de no estar siendo utilizadas últimamente.

Políticas de asignación

Para sistemas con capacidad de multiproceso, la sustitución de páginas se puede clasificar en dos categorías:

- a. *Local*. Para cada proceso se seleccionan sólo marcos asignados a él. El número de marcos de un proceso no cambia. Sólo depende del propio proceso.
- b. *Global*. Permiten que para un proceso se seleccione un marco para sustitución del conjunto de todos los marcos, aunque el marco elegido esté asignado en la actualidad a otro proceso. El número de marcos de un proceso puede cambiar a lo largo del tiempo. Depende del proceso y de la actividad de los otros procesos.

El modelo del conjunto de trabajo

La competición entre los procesos por los espacios de memoria puede llevar a conductas que no se producen si los procesos se ejecutan separadamente. *Catástrofe (thrashing)*: bloqueo de un proceso en espera de intercambios de páginas. Se produce cuando el grado de multiprogramación excede un cierto límite. Se observa un aumento de los intercambios de páginas a la vez que disminuye drásticamente el uso del procesador. Se puede limitar mediante el empleo de algoritmos de sustitución de páginas locales para que otros procesos no entren en catástrofe al perder parte de sus marcos de página. A pesar de ello, como la cola para los dispositivos de páginas es muy grande, el tiempo promedio de servicio para un fallo de página aumenta para todos los procesos, estén en catástrofe o no. Se puede prevenir suministrando a los procesos tantas páginas como necesiten. *Conjunto de trabajo*: Conjunto de páginas utilizadas por un proceso en un momento dado. Cuando un proceso no tiene todo su conjunto de trabajo entra en catástrofe al interrumpirse continuamente por fallos de página. Para evitar la catástrofe, el grado de multiprogramación no debe ser mayor que el que permite que los conjuntos de trabajo de todos los procesos se puedan tener simultáneamente en la memoria. Para determinar que páginas constituyen el conjunto de trabajo de un proceso se hace uso del *Principio de localidad*: las referencias de los programas tienden a agruparse en pequeñas zonas del espacio de direcciones, y estas localizaciones tienden a cambiar sólo intermitentemente. Siguiendo este principio, el conjunto de trabajo es el conjunto de páginas que han sido referenciadas recientemente, y este variará lentamente con el tiempo. En las políticas de sustitución y asignación se debe seguir la siguiente regla genérica: *Ejecutar un proceso solamente si el conjunto de trabajo está por completo en memoria principal y no eliminar una página que es parte del conjunto de trabajo de un proceso.*

Aspectos de diseño para los sistemas de paginación

Tamaño de página

Las páginas se eligen de tamaños que son potencias de dos. El diseñador del SO suele tener capacidad para elegir el tamaño de las páginas, pero no en el caso de las máquinas ya construidas en las que éste viene impuesto. Un tamaño pequeño para las páginas supone un menor desperdicio de la memoria. En promedio, la mitad de la última página de código o de datos queda vacía, produciéndose una fragmentación interna de media página. Por otro lado, si las páginas son pequeñas las tablas de páginas tienen que ser grandes, y como cada proceso tiene su propia copia de la tabla sería recomendable tablas pequeñas, lo que implica páginas grandes. Debe haber un compromiso entre ambas situaciones. Otro problema es el tiempo de lectura/escritura de la página, que está compuesto por un tiempo de latencia y por el tiempo de transferencia. El tiempo de latencia incluye el tiempo de búsqueda y rotación del disco y no depende del tamaño de la página. El tiempo de transferencia sí que depende del tamaño de página, ya que es proporcional a la cantidad de información transferida. En páginas pequeñas hay mayor tiempo de lectura/escritura, y en páginas grandes el tiempo es menor. En lo referente a la relación entre el tamaño de la página y el tamaño de los sectores del disco no se ha logrado una respuesta satisfactoria global a la cuestión del tamaño de las páginas.

Prepaginación

Técnica por la que se traen a la memoria principal de una sola vez todas las páginas que se precisan de un proceso que estaba suspendido o se inicia. El problema es saber si el coste de la prepaginación es menor que el coste de servir fallos de páginas. Suponiendo que p páginas son prepaginadas y que sólo se van a usar una fracción α ($0 \leq \alpha \leq 1$) de ellas, la cuestión es si el coste de prepaginar $(1-\alpha)p$ páginas no usadas es menor que el coste de evitar αp fallos de páginas. Si α es cercano a 1 la prepaginación es rentable, si es cercano a 0, no lo es. El modelo del conjunto de trabajo es útil para realizar la prepaginación.

Frecuencia de fallos de páginas

El modelo del conjunto de trabajo no es el más adecuado para manejar el control de la catástrofe. Este se puede efectuar mejor mediante estrategias que consideren la frecuencia de los fallos de página. Se pueden establecer límites entre los cuales se desea mantener los fallos de páginas. Si se ve que se va a superar el límite, se bloquea un proceso y se reparten sus marcos de página entre los otros procesos.

5. GESTIÓN DEL SISTEMA DE ARCHIVOS

En el almacenamiento de la información a largo plazo hay tres condiciones:

- Se debe poder almacenar una cantidad grande de información.
- La información debe permanecer cuando el proceso termina.
- Debe ser posible que varios procesos tengan acceso concurrente a la información.

La solución es el almacenamiento en un sistema auxiliar (principalmente en discos), en una estructura de archivos. El sistema operativo se puede estudiar desde dos puntos de vista. Los usuarios se preocupan por la forma de nombrar los archivos, las operaciones permitidas, la apariencia del árbol de directorios y otros aspectos del interfaz. Desde el punto de vista de los diseñadores se tiene en cuenta la forma de almacenamiento de archivos y directorios, la administración del espacio en disco, aspectos de eficiencia y confianza al usuario en su utilización.

Archivos

El sistema operativo da una visión uniforme para todos los sistemas de almacenamiento, definiendo una unidad lógica de almacenamiento denominada *archivo*. Es función del sistema operativo el encargarse de asignar el contenido del archivo a espacios en el dispositivo físico.

Un archivo es un conjunto de información relacionada definida por su creador, en general es una serie de bits cuyo significado está definido por su autor y los usuarios. Los archivos son nombrados y referenciados por su nombre (cadena de caracteres alfanuméricos).

En MS-DOS los nombres de los archivos son cadenas de hasta ocho caracteres (alfanuméricos, sin distinguir mayúsculas de minúsculas) y suelen incluir una extensión (tres caracteres después de punto) que indica el tipo de archivo. En Unix se permiten nombres de archivo más largos distinguiéndose mayúsculas de minúsculas.

Los archivos tienen otras propiedades como la fecha y hora de su creación, el nombre o identificador del creador, su longitud, etc. A estas propiedades se denominan atributos y varían de un sistema operativo a otro.

Tipos y estructuras de los archivos

Dependiendo del uso, los archivos tendrán una determinada estructura. El sistema operativo puede tener conocimiento de las distintas estructuras lógicas y considerar diferentes tipos de archivos. Esto permite al sistema operativo dar un mejor servicio. Las desventajas de tener conocimiento de los distintos tipos de archivos por parte del sistema operativo son dos:

- Un mayor tamaño del SO, ya que tendrá que tener el código para soportar cada uno de los tipos de archivos considerados.
- Se tiene una mayor rigidez, ya que sólo se pueden considerar los tipos de archivos definidos por el SO.

En Unix no se definen tipos de archivos, dejando plena libertad al usuario, de esta forma los programas de aplicación deben incluir el código que interprete la estructura adecuada de los archivos. Unix sólo interpreta unos tipos especiales de archivos: los *directorios* y los *archivos especiales* usados para modelar periféricos. El resto son los denominados *archivos regulares* y corresponden a los que tienen información de los usuarios.

Los sistemas operativos como Unix o MS-DOS ven los archivos como una serie de bytes cuyo significado lo dan los programas de los usuarios. El sistema operativo sólo empaqueta y desempaqueta los bytes cuando los escribe o los lee del disco de acuerdo a los bloques físicos del mismo.

El acceso a los archivos puede ser secuencial o arbitrario, los de acceso secuencial son leídos registro a registro. Proviene del almacenamiento en cinta. Hoy en desuso. En los de acceso arbitrario las operaciones de lectura y escritura deben incluir el número del registro al cual se accede directamente. Proviene de la aparición de los discos. Todos los archivos actuales son de este tipo.

Operaciones con los archivos

CREATE (Crear)

Llamada: CREATE (*nombre_archivo, atributos*)

Acciones:

- Buscar si ya existe ese archivo. En ese caso o enviar un mensaje de duplicado, o crear una nueva versión y sobregabar.
- Asignarle una entrada en el directorio.
- Asignarle espacio en el disco.
- Registrar los atributos del archivo en el directorio.

En algunos sistemas esta operación lleva implícita la apertura del archivo.

OPEN (Abrir)

Su propósito es establecer un enlace entre el programa y el archivo. Para ello se trasladan los atributos y la lista de direcciones del archivo, que están en el disco, a la memoria principal, para permitir un rápido acceso en futuras llamadas.

Llamada: *ID_conexión* = OPEN (*nombre_archivo*, *modo_acceso*)

Acciones:

1. Buscar si existe ese archivo. Si no existe enviar un mensaje de error.
2. Verificar el permiso de acceso al archivo. Si no hay permiso, indicar error.
3. Crear un bloque de control de archivo copiando a memoria principal la lista de atributos y direcciones.
4. Crear el identificador de conexión *ID_conexión*.
5. Devolver el identificador de conexión *ID_conexión*.

SEEK (Buscar)

Se cambia la posición del apuntador para señalar al byte o registro cuya dirección lógica se suministra en la llamada.

Llamada: SEEK (*ID_conexión*, *posición_lógica*)

Acciones:

1. Verificar el identificador de conexión *ID_conexión*. Si no se localiza enviar un mensaje de error.
2. Calcular posición adecuada.
3. Actualizar el marcador de archivo.

READ (Leer)

Por lo general la lectura se realiza en la posición lógica actual, y en la llamada se tiene que especificar la cantidad de datos necesarios y proporcionar un buffer para su transmisión. Como se pueden producir diferentes tipos de errores, la función READ indica el resultado de la operación mediante el valor que devuelve en *status*.

Llamada: *status* = READ (*ID_conexión*, *núm_bytes*, *in_buffer*)

Acciones:

1. Verificar el identificador de conexión *ID_conexión*. Si no se localiza enviar un mensaje de error.
2. Verificar archivo abierto para lectura: autorización de acceso.
3. Sincronizar con otros usuarios activos, si hace falta: compartimiento.
4. Calcular número y direcciones de los sectores a leer: correlación.
5. Emitir orden de lectura al controlador del dispositivo.
6. Verificar resultado de lectura del dispositivo.
7. Copiar *núm_bytes* de datos desde el almacenamiento intermedio al *in_buffer*.
8. Actualizar el marcador de archivo.
9. Devolver *status*.

La mayoría de los sistemas requieren una orden OPEN antes de ejecutar READ. Otros llevan implícita la orden OPEN en la orden READ.

WRITE (Escribir)

Llamada: *status* = WRITE (*ID_conexión*, *núm_bytes*, *out_buffer*)

Acciones:

1. Verificar el identificador de conexión *ID_conexión*. Si no se localiza enviar un mensaje de error.
2. Verificar archivo abierto para escritura: autorización de acceso.
3. Sincronizar con otros usuarios activos, si hace falta: compartimiento.
4. Si se amplía el archivo, asignar los bloques requeridos: asignación de espacio.
5. Actualizar directorio si se añaden nuevos bloques.
6. Calcular número y direcciones de los sectores a escribir: correlación.
7. Copiar *núm_bytes* de datos desde el *out_buffer* al almacenamiento intermedio.
8. Emitir orden de escritura al controlador del dispositivo.
9. Verificar resultado de escritura del dispositivo.
10. Actualizar el marcador de archivo.

11. Devolver *status*.

Algunos sistemas tienen una forma restringida de WRITE, con la que sólo pueden añadirse datos al final del archivo (APPEND). La mayoría de los sistemas requieren una orden OPEN antes de ejecutar WRITE. Otros llevan implícita la orden OPEN en la orden WRITE.

CLOSE (Cerrar)

Básicamente deshace el trabajo de OPEN, cerrando las referencias al archivo.

Llamada: CLOSE (*ID_conexión*)

Acciones:

1. Verificar el identificador de conexión *ID_conexión*. Si no se localiza enviar un mensaje de error.
2. Verificar el permiso de acceso al archivo. Si no hay permiso, indicar error.
3. Si la copia de la entrada del directorio guardada en el bloque de control del archivo ha sido actualizada, entonces reescribirla en el disco.
4. Liberar el bloque de control de archivo.
5. Borrar identificador.

DELETE (Borrar)

Cuando se quiere eliminar un archivo se debe realizar esa petición al SO para que realice esa tarea. Hay algunos SO's que mantienen una copia de los archivos borrados hasta que reciben la orden expresa de liberar este espacio.

Llamada: DELETE (*nombre_archivo*)

Acciones:

1. Buscar el nombre del archivo en el directorio. Si no se localiza enviar un mensaje de archivo no encontrado.
2. Verificar permisos. Si no hay permiso, indicar error.
3. Verificar si se está utilizando. Si está abierto enviar mensaje de archivo abierto.
4. Liberar la entrada en el directorio.
5. Liberar el espacio asignado al archivo.

RENAME (Renombrar)

Cambia el nombre del archivo directamente en el directorio.

Lectura y modificación de atributos

Por ejemplo, en Unix el usuario puede modificar los atributos relativos al modo de protección de los archivos.

COPYFILE (Copiar)

Lleva implícita la creación de un nuevo archivo.

Llamada: COPYFILE (*nombre_archivo_origen, nombre_archivo_destino*)

Acciones:

1. Abrir el archivo origen *nombre_archivo_origen*. Si no se localiza enviar un mensaje de archivo no encontrado.
2. Crear el archivo destino *nombre_archivo_destino*, con permisos de escritura. Si no se puede crear (ya existe) enviar un mensaje de archivo ya existente.
3. Abrir el archivo destino.
4. Realizar la copia. Leer bloques del archivo origen *nombre_archivo_origen* y copiarlos en el archivo destino *nombre_archivo_destino*.
5. Cerrar el archivo origen *nombre_archivo_origen*.
6. Cerrar el archivo destino *nombre_archivo_destino*.

Directorios de archivos

Los directorios son, básicamente, tablas simbólicas de archivos. Una entrada de directorio puede contener la siguiente información:

- a. Nombre, tipo y número de versión del archivo.
- b. Puntero de acceso al archivo, dirección de comienzo en el disco.
- c. Lista de atributos como tamaño, fecha de creación, estructura, fecha última copia de seguridad, dueño, fecha última modificación o referencia, modos de protección etc.

En muchos sistemas la tabla del directorio está dividida en dos. En una sólo se mantienen los nombres de los archivos con un número de identificación, el cual da acceso a la otra que es donde se tiene el puntero de acceso al archivo y la lista de atributos. La separación de la tabla de nombres de archivos de la de descripción agiliza la gestión de los enlaces, la generación de alias y los homónimos.

El número de directorios varía de un sistema a otro. Uno de los diseños más sencillos es el *directorio de nivel único*: un único directorio contiene todos los archivos del sistema o del volumen. Pero presenta muchos inconvenientes, sobre todo en sistemas multiusuario: duplicación de nombres, uso compartido, etc. Una mejora se obtiene al considerar un directorio por

usuario. Así se elimina el conflicto entre ellos. Pero los usuarios con muchos archivos siguen teniendo un gran problema de organización de los mismos. La solución es permitir una jerarquización total, es decir, crear un *árbol de directorios*. De esta forma, cada usuario dispone de su propia estructura jerárquica con tantos subdirectorios como necesite. Normalmente, cada usuario tiene su directorio inicial, es decir, aquel en el que se inicia la sesión del usuario.

Hay dos tipos de nombres de los caminos:

- a. *completos*: comienzan en el directorio raíz y bajan por el árbol de directorios hasta el archivo indicado.
- b. *relativos*: comienzan en el directorio actual y bajan por el árbol de directorios hasta el archivo indicado.

La mayoría de SO's actuales con estructura de directorios jerárquicos tienen dos entradas especiales para cada directorio: "." (punto): es una entrada para el propio directorio (un puntero a sí mismo). ".." (punto-punto): es una entrada para el directorio padre (el que está por encima en la jerarquía).

Otro problema es dónde y cómo ubicar los directorios. Se adoptó la solución de tenerlos en el propio disco pues pueden ser muy grandes para tenerlos en memoria principal. Para uniformizar las operaciones en disco, es habitual considerar los directorios como archivos. De esta forma los directorios son archivos que tienen una lista de todos los archivos. El problema, entonces, es localizar el directorio raíz al arrancar el sistema. La solución es situar el directorio raíz en una dirección conocida por el volumen desde el que se arranca el sistema.

Operaciones con directorios

Varían mucho de unos sistemas a otros. La siguiente lista se refiere a las operaciones más importantes bajo sistemas como Unix (Unix considera a los directorios como archivos con unas características especiales).

MAKEDIR (Crear directorio)

Crea un subdirectorio en el directorio actual. Estará vacío salvo por sus entradas por defecto, "." y "..".

REMOVEDIR (Borrar directorio)

En principio se supone que está vacío. Si no lo está se pueden hacer dos cosas:

- No permitir borrar el directorio. Esta es la opción por defecto en Unix. Se deben borrar previamente todos los archivos y subdirectorios que contiene.
- Suponer que se sabe que se borrarán también sus archivos y subdirectorios. Es una opción más cómoda pero más peligrosa para el usuario (puede borrar accidentalmente un subdirectorio o un archivo que no deseaba eliminar).

OPENDIR (Abrir directorio) y **CLOSEDIR** (Cerrar directorio)

Igual que con los archivos, cuando se quiere operar con los directorios hay que abrirlos. En muchos sistemas esto supone copiar las tablas del directorio en la memoria principal de forma que se agilice la gestión. Cuando se termina de utilizar el directorio se cierra, actualizándose las tablas del mismo.

READDIR (Leer directorio)

La orden *ls* de Unix da un listado de todas las entradas del directorio actual. Por defecto sólo se da el nombre de los archivos y subdirectorios, pero puede dar listados extendidos con el contenido completo de las entradas del directorio especificado.

RENAMEDIR (Cambiar de nombre)

Los directorios se pueden cambiar de nombre con una llamada al sistema, al igual que con los archivos.

LINK (Enlazar)

Es una forma de hacer que un archivo o subdirectorio aparezca en varios directorios. Es una forma de hacer que un archivo o directorio lo compartan dos usuarios, pues se pueden crear enlaces desde cada uno de los directorios de los usuarios. En algunos sistemas esta función es la que corresponde a la creación de *alias*, que permite asociar a un archivo varios nombres.

UNLINK (Desenlazar)

Es la operación contraria a LINK. Si sólo hay un camino especificado para el archivo, UNLINK lo eliminará. En Unix, la llamada REMOVEDIR es, de hecho, un UNLINK.

Realización del sistema de archivos

Desde el punto de vista del usuario, un archivo es un tipo de dato abstracto que se puede tratar sin preocuparse de su implementación, desde la óptica del sistema operativo el problema es cómo diseñar los archivos y cómo asignarles espacio de forma que la memoria del disco sea utilizada de una forma eficiente y se acceda rápidamente a los archivos. Esto supone elegir una política de asignación de espacio a los archivos y de gestión del espacio libre del disco. En general, existen dos estrategias para almacenar un archivo de n bytes:

1. Asignar n bytes consecutivos del espacio del disco.
2. Dividir el archivo en bloques que no necesitan estar adyacentes.

En la asignación del espacio del disco, los tres métodos más usados son:

- a. Asignación contigua.
- b. Mediante listas enlazadas.
- c. Mediante índices.

La mayoría de los SO's tienen alguno de estos métodos o una combinación de ellos.

Gestión del espacio libre

Una cuestión clave en la gestión del sistema de archivos es mantener una lista del espacio libre en el disco. Para mantener esta lista de espacio libre se usan principalmente dos métodos:

- a. Tener un mapa de bits de los bloques libres. Un disco con n bloques necesitará un mapa de bits con n bits. Los bloques libres se representarán con un 1 y los ocupados con un 0 (o viceversa).
- b. Usar una lista enlazada de los bloques libres, manteniendo un puntero al primer bloque libre. Este esquema no es muy eficiente, puesto que para recorrer la lista hay que leer todos los bloques.

Una modificación de este método es utilizar una lista enlazada de bloques, en la que cada bloque contiene tantos números de bloques libres como pueda.

El método del mapa de bits ocupa menos espacio, ya que sólo usa 1 bit por cada bloque. Si además existe suficiente espacio en la memoria principal para mantener el mapa de bits, este método es preferible. Sin embargo, cuando el disco está casi lleno podría ser más rentable el método de las listas enlazadas.

Otra cuestión clave es la elección del tamaño de los bloques para la asignación: los bloques grandes tienen el inconveniente de desaprovechar el espacio remanente; los pequeños que se precisarán muchos bloques para un solo archivo. Por lo tanto, la elección del tamaño de los bloques requiere hacer un estudio previo de cuan grandes en promedio van a ser los archivos en ese sistema. Un compromiso habitual es elegir los bloques de 512 bytes, 1 Kbyte o 2 Kbytes. Si se utiliza un tamaño de 1 Kbyte en un disco cuyos sectores son de 512 bytes, entonces el sistema de archivo siempre leerá y escribirá dos sectores consecutivos que considerará como una unidad indivisible.

Método de asignación contigua

Este esquema requiere que cada archivo ocupe un conjunto de direcciones contiguas en el disco. La asignación contigua de un archivo se define por la dirección del primer bloque y la longitud del archivo. La dificultad con este método es asignarle el espacio correcto cuando se crea el archivo. Se tiene que buscar un hueco donde hayan el número de bloques contiguos necesarios. Las soluciones son del tipo primero en ajustarse o del mejor en ajustarse. Pero estos sistemas sufren de fragmentación externa. Además se debe conocer el tamaño exacto del archivo en el momento de su creación. Los programas que crean archivos de forma dinámica van a tener muchas dificultades, también se produce mucha fragmentación externa en el disco. La solución es la compactación de disco, pero ésta es una operación costosa, aunque se podría hacer en periodos de inactividad (por ejemplo durante las noches).

Método de asignación mediante listas enlazadas

Una solución a los problemas de la asignación contigua es mantener los archivos como una lista enlazada de bloques de disco. Unos pocos bytes del comienzo de los bloques se usan como puntero al siguiente bloque; el resto del bloque contiene los datos del archivo. En el directorio se indica el bloque inicial y el bloque final del archivo. Escribir en el archivo supone coger uno de los bloques libres y añadirlo al final de la lista. Para leer el archivo sólo hace falta seguir los punteros de bloque en bloque. Esta técnica no causa fragmentación externa, tampoco se tiene que declarar el tamaño del archivo cuando se crea. Además, los archivos pueden crecer sin ningún problema. Tiene el inconveniente de que el acceso aleatorio a un archivo es extremadamente lento.

Método de asignación mediante indexación

Se colocan los índices a los bloques del archivo en una *tabla de índices*, es decir, se usa un método de asignación mediante indexación. En el directorio se encuentra la dirección del bloque que contiene el índice de los bloques del archivo de forma que todo el bloque está disponible para los datos. No se debe recorrer secuencialmente todos los bloques del archivo para acceder a la información de un bloque en concreto. Su inconveniente es la pérdida de espacio ya que cada archivo consume un bloque completo para los índices. La solución a este problema consiste en utilizar un sistema de indexación de varios niveles: el bloque de índices es pequeño y contiene referencias no a los bloques del archivo sino a otros bloques de índices, y así sucesivamente. Dependiendo del tamaño del archivo el sistema de niveles tiene más subniveles o menos (este es el método utilizado por Unix).

Algunos aspectos de diseño

Un sistema de archivos presenta varios problemas de diseño, en primer lugar hay que definir como verá el usuario el sistema de archivos. Luego se deberán crear los algoritmos y las estructuras de datos que relacionen el sistema de archivos lógicos con el dispositivo físico de almacenamiento.

En segundo lugar el sistema de archivos se pueden considerar compuesto por varios niveles, es decir, tiene un diseño por capas. Cada nivel en el diseño usa las posibilidades de los niveles inferiores para crear nuevas características que usarán los niveles superiores. El nivel más bajo consiste en los dispositivos físicos de control de las transferencias de información

entre la memoria y el disco. El siguiente nivel es el sistema básico de archivos, el cual usa las características del nivel anterior para leer y escribir bloques al y desde el disco. La indexación de los bloques que tiene que leer o escribir el sistema básico de archivos lo hace un módulo de organización de archivos, que conoce el tipo de ubicación usada por los archivos y la situación de los mismos.

La estructura de directorios la conoce el sistema de archivos lógicos. Los directorios se pueden tratar como archivos de forma que el módulo de organización de estos puede leer y escribir los directorios cuando se producen peticiones del sistema de archivos lógicos. Por último se tendrán los programas de aplicaciones.

Otro aspecto es el control de acceso, verificando las admisiones a los archivos y a los directorios para que estos sean correctos. Así, cuando un archivo se abre para leer, el SO debe impedir la escritura en él.

Compartición de archivos

La compartición de archivos se puede realizar de varias formas. Una de las más comunes, que se utiliza en Unix, consiste en tener una pequeña estructura de datos, el nodo-i, asociada al propio archivo, de forma que los directorios apuntarían al nodo-i correspondiente. Ese nodo-i contiene, además, el nombre del propietario original (antes de la compartición) del archivo y un contador con el número de enlaces existentes.

El problema surge cuando el propietario original desea eliminar un archivo compartido. Si se elimina sin más, los directorios que lo compartían tendrán una entrada a un nodo-i no válido. La solución consiste en dejar intacto el nodo-i y el archivo, pero se decrementa el contador. El propietario del archivo sigue siendo el mismo. Este problema se puede resolver mediante *enlaces simbólicos*. En esta situación se crea un nuevo archivo que sólo contiene la ruta completa de acceso al archivo al que se quiere enlazar. Sólo el propietario verdadero tiene un apuntador al nodo-i. Así, si el propietario elimina el archivo este se destruye, y los posteriores intentos de otros usuarios de acceder a él fracasarán. Este sistema tiene como ventaja que se puede utilizar para enlazar archivos de otras máquinas, proporcionando sólo la dirección de la red de la máquina donde reside el archivo además de la ruta de acceso en esa máquina. Su principal inconveniente es su alto coste ya que se debe leer el archivo que contiene la ruta de acceso, analizarla y seguirla hasta alcanzar el nodo-i, lo cual supone un considerable número de accesos adicionales al disco. Además del coste en espacio para los enlaces. Otra desventaja de los archivos compartidos, en general, es que la restauración de una copia de back-up puede duplicar archivos si no se ha preparado adecuadamente.

Cachés de disco

El cuello de botella en muchas de las aplicaciones de los computadores está en los tiempos de acceso al disco, que es del orden de cien mil vez más lento que a la memoria principal. Una de las técnicas más comunes para mejorar este acceso es la utilización de *cachés de disco*. La idea, similar a las cachés de memoria, es mantener un depósito de copias de bloques del disco en la memoria principal, de forma que permita eliminar accesos repetitivos al disco.

Entre los distintos algoritmos para la administración de la caché, uno de los más comunes es verificar todas las solicitudes de lectura/escritura para saber si el bloque solicitado se encuentra en la caché. En caso afirmativo se evita el acceso al disco, sino se debe leer el bloque del disco y transferirlo a la caché. Para sustituir bloques se usan algoritmos similares a los de memoria: FIFO, LRU, etc.

El problema principal es evitar inconsistencias entre la caché y el disco cuando un bloque en la caché se ha modificado y el sistema se apaga accidentalmente o se "cae" ya que podría quedar corrompido todo el sistema de archivos. En Unix se hace un vaciado de la caché a intervalos periódicos mediante una llamada al sistema (sync), que obliga a todos los bloques modificados a actualizarse en el disco. Por lo tanto, un sistema Unix nunca se debe apagar sin antes utilizar el procedimiento shutdown para vaciar las cachés. Debido a ello los sistemas Unix suelen ejecutar al arrancar la utilidad fsck para la verificación del sistema de archivos. Unix se diseñó como un sistema en el que el sistema de archivos reside en un disco no extraíble. En disco extraíble no debe ser extraído hasta ejecutar un sync.

MS-DOS usa cachés de escritura directa, es decir, cada bloque que se modifica es reescrito inmediatamente en el disco. Esto disminuye el rendimiento. MS-DOS se diseñó como un sistema que debía soportar un sistema de archivos no necesariamente residente en discos no extraíbles.

Los inconvenientes del uso de caché es que supone una gestión más compleja en el acceso a los archivos de los discos y presenta el peligro potencial de las inconsistencias o corrupciones indicadas.

Sus ventajas son que mejora el tiempo efectivo de acceso al disco y el tiempo de respuesta de las aplicaciones, al utilizar la escritura retardada en disco las aplicaciones no tienen por qué esperar la terminación de dicha escritura, la eliminación de algunos acceso al disco. Si además el sistema es suficientemente inteligente se puede conseguir que los archivos temporales, no sean nunca escritos en el disco y permanezcan en la caché hasta su eliminación. Así en Unix se

llegan a eliminar hasta un 85% de los accesos al disco. En la utilización de los discos compartidos en red, también supone una gran reducción de la carga en el servidor de archivos y en la red, dado que los archivos de los discos compartidos pueden ser capturados en caché.

Seguridad y protección

El término seguridad se suele referir al problema general y el término mecanismo de protección a los procedimientos específicos utilizados por el sistema operativo para asegurar la información del computador. Los orígenes de los problemas de seguridad pueden ser producidos por actos fortuitos debidos a causas ajenas al sistema informático, como incendios, apagones de luz, etc. por averías en el propio computador o errores de los programas o por errores humanos o actos malintencionados.

Integridad del sistema de archivos

Los problemas en el sistema de archivo pueden venir por el uso de bloques del disco que están en mal estado. Este problema se puede solventar conociendo cuales son los bloques defectuosos y hacer que el SO evite su uso.

Las inconsistencias en el sistema de archivos pueden ocurrir por otras muchas causas, por ejemplo, si el sistema falla en la mitad de una operación de lectura, modificación y escritura de un bloque. Si además el bloque que se ha modificado, y que no ha dado tiempo a escribirlo, es un bloque con nodos-i, directorios o la lista de bloques libres el problema puede ser crítico.

Problemas usuales de inconsistencia que se pueden presentar son:

- a. Si un bloque aparece en la lista de bloques usados y en la de libres, se elimina de la de libres.
- b. Si un bloque no aparece en ninguna lista, se añade a la lista de bloques libres.
- c. Si un bloque está repetido en la lista de bloques libres, se reconstruye la lista de bloques libres.
- d. Si un bloque está asignado a dos o más archivos se produce la peor situación. Hay dos casos:
 - Si se elimina uno de los archivos entonces se presenta la situación *a*.
 - Si se eliminan los dos archivos entonces se presenta la situación *c*.

La solución es que se asigne un bloque libre a un archivo y se copie el contenido del bloque que estaba asignado a los dos; de esta forma no se tendrán bloques repetidos en las listas de cada archivo, aunque seguramente la información de los archivos no será consistente.

Los Sistemas Operativos tienen utilidades para tratar estos problemas. Unix, por ejemplo, posee la utilidad *fsck* que se puede ejecutar cada vez que el administrador del sistema lo crea conveniente, pero, además, si cuando se arranca el sistema se comprueba que no fue apagado adecuadamente, se ejecuta automáticamente para detectar las inconsistencias que pudieran haber ocurrido por un mal apagado y las corrige. Cuando el deterioro del sistema de archivos es irreparable es necesario disponer de copias de seguridad a partir de las cuales poder restaurarlo.

La forma más fácil de crear las copias de seguridad es realizar volcados periódicos de todo el sistema de archivos. Otra forma es realizar *volcados incrementales* en los que sólo se copian los archivos modificados desde la última copia. Para ello, por ejemplo MS-DOS, dispone del llamado *bit de biblioteca* que se pone a 1 cuando se modifica un archivo y a 0 cuando se realiza la copia de seguridad. Esta fórmula debe ir acompañada de volcados completos periódicos. La principal desventaja de los volcados incrementales es la cantidad de datos que se generan y la complejidad del procedimiento de restauración.

Ataques a la integridad y seguridad del sistema de archivos

Un problema grave es el de los intrusos que intentan acceder al sistema de archivos por curiosidad o para corromperlo o beneficiarse económicamente. El objetivo de la seguridad es prevenir y eliminar estas amenazas. Un sistema seguro debe *mantener la integridad* (los datos deben ser correctos), *la disponibilidad* y *la privacidad* de la información. Esto supone la protección frente a *modificaciones no autorizadas* y a la *modificación no detectada de datos*, así como la *resistencia a la penetración*.

Las formas más conocidas de penetración en un sistema informático son:

- a. La utilización por parte de un intruso de la cuenta de un usuario legítimo. Puede conseguirlo accediendo desde un terminal a una sesión abierta del usuario o obteniendo la contraseña por el medio que sea.
- b. La ejecución de programas llamados *caballos de Troya*, los cuales ocultan parte de su funcionalidad, frecuentemente destinada a obtener datos o derechos de acceso del usuario. Por ejemplo, un intruso puede crear un programa falso de login idéntico al del sistema de forma que el usuario escribirá su "login" y contraseña, la cual será utilizada posteriormente por el intruso para acceder a sus archivos y programas.
- c. La propagación de *gusanos* y *virus informáticos*. Los gusanos son programas en si mismos, causan graves problemas al sistema ya que carga en exceso al computador, usando para su propagación recursos desproporcionados de procesamiento y comunicación, con lo que el sistema puede denegar servicios a usuarios legítimos. Los virus informáticos son un trozo de código de un programa, que infectará a otros programas copiándose. Por lo general también realiza actividades dañinas, como eliminar archivos o corromper los bloques de arranque del disco. Normalmente el virus vendrá en el código de un programa.

Al arrancar el programa, el virus examina todos los programas del disco, si encuentra uno sin infectar, lo infecta añadiendo su código al final del archivo y sustituyendo la primera instrucción del programa por un salto al principio del código del virus

- d. La inspección del sistema de archivos. Sistemas como Unix permiten, por defecto, a todos los usuarios leer los archivos. Un usuario malintencionado podría acceder al archivo de contraseñas, copiarlo y utilizar técnicas de análisis criptográfico y métodos de prueba y error para descifrar las contraseñas, además en estos archivos suele haber información relativa a los usuarios.

Principios de diseño de sistemas seguros

Saltzer y Schroeder (1975) identificaron varios principios generales que se pueden utilizar como guía para el diseño de sistemas seguros:

1. El diseño del sistema debe ser público. No se puede confiar en la ignorancia de los atacantes.
2. El estado predefinido es el de no acceso. Los derechos de acceso se adquieren sólo con permiso explícito.
3. Verificar la autorización actual. Esta verificación de autorización no debe ser abandonada por el sistema, ya que pueden producirse situaciones como la de que un usuario abra un archivo y lo mantenga abierto durante días, aunque cambien las autorizaciones de uso de este archivo.
4. Mínimos privilegios. Cada proceso debe utilizar el mínimo número de privilegios para completar su tarea.
5. Mecanismos simples e integrados. Mantener el diseño tan sencillo como sea posible facilita la verificación y corrección de las implementaciones. Además, para que el sistema sea verdaderamente seguro, el mecanismo debe estar integrado hasta las capas más bajas del sistema.
6. Psicológicamente aceptable. El mecanismo debe ser fácil de usar de forma que sea aplicado correctamente y no sea rechazado por los usuarios.

Identificación de los usuarios

Muchos sistemas de seguridad se basan en la suposición de que el sistema conoce al usuario. El problema en este caso es la identificación del mismo. Este problema se suele llamar de validación y se basa en tres puntos (o una combinación de ellos):

1. Posesión de algún secreto, algo conocido por el usuario. Una contraseña que le de acceso al sistema. Por lo general, el sistema pide una contraseña al usuario. Esta contraseña se suele hallar cifrada en una tabla en un archivo del sistema. Posee muchos puntos débiles. Los usuarios suelen elegir contraseñas fáciles de recordar y por lo tanto fáciles de violar. Algunos computadores afrontan este problema pidiendo a los usuarios que periódicamente cambien la contraseña o limitan el número de intentos de acceso.
2. Posesión de un artefacto, algo que al poseerlo el usuario le permite acceder al sistema. Por ejemplo, tarjetas magnéticas o llaves físicas. En este caso, junto a los terminales suele haber un lector de tarjetas o de distintivos. Este método suele estar asociado a una contraseña (terminales bancarios). Otras variantes son las tarjetas inteligentes, que mantienen la contraseña del usuario secreta para el sistema, ya que está almacenada en la propia tarjeta. Esto hace más difícil descubrir las contraseñas.
3. Uso de alguna característica fisiológica o de comportamiento del usuario. Se pueden catalogar en dos grupos *fisiológicas* y *de comportamiento*. Las fisiológicas pueden ser huellas dactilares o vocales, características faciales o geometría de la mano (por ejemplo, dispositivos para medir la longitud de los dedos). Las de comportamiento pueden ser el análisis de firmas o los patrones de voz.

La efectividad de las técnicas de identificación se hace en función de la tasa de falsas aceptaciones y de falsos rechazos.

Mecanismos de protección y control de acceso

Es importante distinguir entre *política de protección* que dice qué se hará y *mecanismo de protección* que dice cómo se hará algo.

Dominios de protección

Para analizar los mecanismos de protección es útil el concepto de *dominio de protección*, cada proceso trabaja dentro de su dominio, el cual especifica los recursos a los cuales puede acceder. Cada dominio define un conjunto de objetos y las operaciones que se les pueden aplicar. Un dominio es un conjunto de derechos de acceso, cada uno de los cuales está formado por un par de la forma:

<nombre del objeto, conjunto de sus derechos>

Por ejemplo, en Unix el dominio de un proceso está definido por el identificador del usuario (uid) y del grupo (gid). Para un uid y gid dados se puede elaborar una lista completa de todos los objetos a los que puede tener acceso y el tipo de acceso, es decir, una lista de archivos, dispositivos, etc., con sus correspondientes permisos de acceso.

Matriz de acceso

Las relaciones entre dominios y objetos se pueden representar de forma abstracta mediante una matriz denominada *matriz de acceso*. Esta matriz tiene en cada fila un dominio y en cada columna un objeto. Generalmente, la matriz de acceso es una matriz dispersa, con muchos huecos vacíos. Por ello, una forma sencilla de realizarla es mediante una tabla global, consistente en tripletas ordenadas (dominio, objeto, derechos).

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

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

Lista de accesos

A cada objeto se le asocia una lista ordenada con todos los dominios que pueden tener acceso al objeto y la forma de dicho acceso. El principal inconveniente de las listas de accesos es el retardo que se provoca con la búsqueda para verificar la autoridad de un sujeto para acceder al objeto solicitado.

Algunos sistemas dividen a los usuarios en grupos y sólo almacenan los derechos de acceso de los grupos. Este esquema ahorra almacenamiento y agiliza el procesamiento reduciendo la flexibilidad y limitando el número de dominios. Unix usa este tipo de esquema: las listas están reducidas a tres entradas por archivo, una para el propietario, otra para el grupo y otra para todos los usuarios.

Lista de capacidades

A cada dominio se le asocia una lista de objetos a los cuales puede tener acceso, junto con una indicación de las operaciones permitidas sobre cada objeto. La protección se basa en que nunca se permite que una capacidad se mueva al espacio de direcciones accesibles por un proceso de un usuario.

Para realizar esta operación hay que distinguir las capacidades de otros objetos. Para ello se puede asociar a cada objeto una etiqueta que indica si es una capacidad o un dato accesible. Los programas de aplicaciones no deben tener acceso a estas etiquetas (se consigue mediante hardware o firmware). También se puede dividir el espacio de direcciones asociado a un programa en dos partes. Una parte es accesible al programa y contiene sus datos e instrucciones normales; la otra parte contiene la lista de capacidades y sólo puede acceder a ella el Sistema Operativo. Para esta estrategia es conveniente disponer de un espacio segmentado de memoria.

Tiene como inconveniente la revocación de acceso a un objeto. Una forma de eliminar este problema es hacer que las capacidades apunten a un objeto indirecto en vez de al objeto real. La forma de invalidar la capacidad es romper el enlace entre el objeto indirecto y el real.

6. GESTIÓN DEL SISTEMA DE ENTRADA/SALIDA

Una de las principales funciones del SO es la de controlar todos los dispositivos de entrada/salida, también llamados periféricos. Los periféricos se pueden clasificar en tres categorías:

1. *Adaptados al usuario.* Apropriados para comunicar información al usuario.
2. *Adaptados a la máquina.* Permiten comunicarse con el sistema.
3. *De comunicación.* Preparados para transmitir información a dispositivos remotos que pueden estar adaptados al usuario, a la máquina o ser otro computador.

Los diferentes periféricos se pueden clasificar por:

- a. Velocidad de transferencia.
- b. La utilización a que se dedica el periférico.
- c. La unidad de transferencia.
 - Orientados a bloques. Los datos se transfieren en bloques. Los bloques normalmente son de tamaño fijo. Es posible leer o escribir en un bloque de forma independiente de los demás. Los datos se referencian o direccionan dando el número del bloque que se desea transferir.
 - Orientados a caracteres. Los datos se transfieren como una cadena de caracteres. No están sujetos a una estructura de bloques. No se pueden usar direcciones. No es posible realizar operaciones de búsqueda.
- d. La complejidad del controlador del dispositivo.
- e. Las condiciones de error.

Controlador de E/S

Es el módulo del computador responsable del control de uno o más dispositivos externos y del intercambio de datos entre dichos periféricos y la memoria principal o con los registros de la UCP. Por esta razón, el controlador de E/S debe poseer una interfaz interna al computador (para su conexión con la UCP y con la memoria principal), y una interfaz externa al computador (para su conexión con el dispositivo externo).

Funciones del controlador de E/S

Control y temporización: Los recursos internos del computador se comparten por una serie de actividades entre las que se incluye la E/S de datos. Debido a esto la E/S requiere un mecanismo de control y temporización que coordine el intercambio de información entre los recursos internos y los dispositivos externos. Por ejemplo, el control de transferencia de datos entre un dispositivo externo y la UCP necesita la siguiente secuencia de pasos:

1. La UCP pide al controlador de E/S que compruebe el estado del dispositivo al que está conectado.
2. El controlador de E/S devuelve el estado del dispositivo.
3. Si el dispositivo está operativo y preparado para transmitir, la UCP solicita la transferencia del dato mediante una orden al controlador de E/S.
4. El controlador de E/S obtiene el dato del dispositivo externo.
5. El dato se transfiere desde el controlador de E/S a la UCP.

La Comunicación con la UCP requiere:

- a. Decodificación de la orden recibida de la UCP a través del bus de control.
- b. Datos: El intercambio de datos entre la UCP y el controlador se realiza a través del bus de datos.
- c. Información sobre el estado: A causa de la lentitud de los periféricos es importante conocer el estado del controlador de E/S. De estas situaciones se informa con una "señal indicativa del estado" del controlador de E/S.
- d. Reconocimiento de la dirección: El controlador de E/S reconoce una dirección única para cada uno de los periféricos que controla.

Comunicación con el dispositivo externo: Comprende órdenes, información del dispositivo y datos.

Almacenamiento temporal de datos: Surge por las diferentes velocidades de transferencia de datos que tienen los distintos dispositivos. El almacenamiento temporal en el controlador de E/S sirve para adecuar las diferencias de velocidades entre la interfaz interna con el computador (conexión a la memoria principal y a la UCP) y la interfaz externa (conexión con el dispositivo).

Detección de errores: El controlador de E/S es a menudo responsable de la detección de errores, y de informar a la UCP cuando ocurren. Una clase de errores incluye anomalías mecánicas y eléctricas transmitidas por el propio dispositivo. Otra clase consiste en cambios en la secuencia de los bits que se transmiten desde el dispositivo al controlador de E/S. Con frecuencia se utiliza una codificación de la información que permite detectar errores simples en la transmisión: el bit de paridad.

Estructura del controlador de E/S

Los controladores de E/S varían considerablemente en complejidad y en el número de dispositivos externos que pueden controlar. El controlador de E/S se conecta con el resto del computador a través del bus de sistema. Los datos que se transfieren al controlador o desde el controlador se almacenan temporalmente en uno o más registros de datos. También puede haber más de un registro de estado que proporcione la información del estado actual. El registro de estado a veces funciona como un registro de control que acepta información proveniente de la UCP. A este conjunto de registros genéricamente se les suele llamar *puertos del controlador*. La lógica que hay dentro del módulo interacciona con la UCP mediante un conjunto de líneas de control. Estas líneas las utiliza la UCP para enviar órdenes al controlador de E/S. Algunas de las líneas de control son utilizadas por el controlador de E/S (por ejemplo, para cuestiones de arbitraje o informar del estado en que se encuentra). El controlador debe ser capaz de reconocer y generar las direcciones asociadas con los dispositivos que controla. Cada controlador tiene asociada una única dirección, o un conjunto de ellas si controla a más de un dispositivo externo. Asimismo, el controlador de E/S contiene la lógica específica para la interfaz con cada periférico que controla.

Estructura del sistema de E/S

Hay tres maneras de utilizar el bus para interconectar la UCP con la memoria y con la unidad de E/S:

1. Utilizar dos buses independientes, uno para la memoria y otro para el sistema de E/S. Esta estrategia se emplea en aquellos computadores que, además de la UCP, disponen de un procesador de E/S (PE/S). La memoria se comunica tanto con la UCP como con el PE/S por el bus de sistema. El PE/S también se comunica con los dispositivos de E/S a través de un bus de E/S independiente con sus propias líneas de datos, dirección y control, al que se conectan los controladores de E/S. Otra denominación del PE/S es la de *canal*.
2. Utilizar un bus común para la memoria y el sistema de E/S, pero con líneas de control independientes para cada uno. Es lo que se conoce como *E/S aislada*.
3. Utilizar un único bus con líneas de control también comunes. Es lo que se conoce como *E/S localizada en memoria o mapeada*.

Mecanismos básicos de E/S

E/S Controlada por Programa

En esta técnica los datos se intercambian entre la UCP y el controlador de E/S. La UCP ejecuta un programa que tiene el control directo de la operación de E/S e incluye la comprobación del estado del dispositivo, el envío de una orden de lectura o escritura y la transferencia del dato. Cuando la UCP emite una orden al controlador de E/S debe esperar hasta que finalice la operación de E/S. Si la UCP es más rápida que el controlador de E/S se malgasta tiempo de la UCP.

E/S por Interrupciones

La UCP envía una orden de E/S y continúa ejecutando otras instrucciones hasta que es interrumpida por el controlador de E/S, cuando éste ha finalizado su trabajo.

Acceso directo a memoria (DMA)

El controlador de E/S y la memoria intercambian datos directamente sin la intervención de la UCP.

	Sin interrupciones	Con interrupciones
Transferencia de E/S a memoria a través de la UCP	E/S controlada por programa	E/S por interrupciones
Transferencia de E/S a memoria directa		Acceso directo a memoria (DMA)

E/S CONTROLADA POR PROGRAMA

Cuando se emplea esta técnica es el computador quien “adapta” su velocidad de trabajo a la del periférico. Si la UCP está ejecutando un programa y encuentra una instrucción de E/S, envía una orden al controlador de E/S adecuado. Este controlador realiza la acción pedida y a continuación modifica el contenido de su registro de estado RE y no efectúa ninguna acción más para comunicárselo a la UCP. Así pues, es responsabilidad del procesador el comprobar periódicamente el estado del controlador de E/S hasta que detecta que ha finalizado la operación.

Órdenes de E/S

Para ejecutar una instrucción de E/S, la UCP envía una orden de E/S y una dirección que especifica el controlador y el periférico en particular. La UCP puede enviar al controlador cuatro tipos de órdenes:

- a. *Órdenes de control*: Se utilizan para activar un periférico y decirle qué hacer. Estas órdenes están adaptadas a cada tipo particular de periférico.
- b. *Órdenes de comprobación*: Se utilizan para verificar diferentes condiciones de estado asociadas con un controlador de E/S y sus periféricos.
- c. *Órdenes de lectura*: Originan que el controlador de E/S obtenga un dato del periférico y lo coloque en un registro interno (registro de datos RD). La UCP puede entonces conseguir el dato pidiendo al controlador de E/S que lo coloque sobre el bus de datos.
- d. *Órdenes de escritura*: Realizan el proceso inverso de las órdenes de lectura y hacen que el controlador de E/S tome un dato del bus de datos y a continuación lo transmita al periférico.

Después de efectuar la transferencia de un dato, el computador permanece en un bucle de espera hasta que el periférico está preparado para realizar la siguiente transferencia. El periférico indica su disponibilidad mediante los bits de su registro de estado. El computador no realiza ningún trabajo útil mientras permanece en el bucle de espera.

Instrucciones de E/S

En la E/S controlada por programa, hay una correspondencia muy estrecha entre las instrucciones de E/S que la UCP recibe de la memoria y las órdenes de E/S que la UCP envía al controlador de E/S para su ejecución, las instrucciones se transforman fácilmente en órdenes de E/S, y a menudo hay una relación biunívoca entre ambas. La forma de la instrucción depende de la manera en que se direccionan los dispositivos externos. Cada dispositivo tiene una dirección única. Cuando la UCP emite una orden de E/S, esta orden contiene la dirección de periférico deseado y los controladores de E/S deben interpretar las líneas de dirección para determinar si la orden les compete.

Cuando la UCP, la memoria principal y la unidad de E/S comparten un bus común, la forma de hacer el direccionamiento difiere según que la E/S esté localizada en memoria o esté aislada:

E/S localizada en memoria

- Hay un único espacio de direcciones para las posiciones de memoria y los dispositivos de E/S.
- La UCP trata los registros de datos y de estados de los controladores de E/S como posiciones de memoria y utiliza las mismas instrucciones máquina para acceder tanto a la memoria como a los periféricos.
- Sólo se necesita un conjunto de señales de lectura y escritura.
- Cada controlador de E/S se organiza como un conjunto de registros que corresponden a señales de lectura y escritura en el espacio normal de direcciones.
- Normalmente se reserva un segmento del espacio total de direcciones para los registros internos de los controladores de E/S, aunque pueden estar localizados en cualquier dirección mientras no existan palabras de memoria que correspondan a la misma dirección.
- Los computadores que poseen E/S localizada en memoria pueden usar instrucciones del tipo referencia a memoria para acceder a datos de E/S. Esto permite utilizar el mismo repertorio de instrucciones para E/S que para referencias a memoria. La ventaja es, que las instrucciones de *cargar* y *almacenar* que se emplean para leer y escribir de memoria se pueden usar también para la E/S de datos de los registros del controlador de E/S. En un computador normalmente existen muchas más instrucciones de referencia a memoria que de E/S.

E/S aislada

- El bus de sistema dispone, además de las líneas de control de lectura y escritura en memoria, de líneas de control específicas de E/S para acceder a los periféricos. De esta forma, la línea de control especifica si la dirección se refiere a una posición de memoria o a un periférico.
- El rango completo de direcciones está disponible para ambos.

El mecanismo de E/S controlada por programa presente una serie de inconvenientes:

1. Pérdida de tiempo en el bucle de espera.
2. Si existen programas que tienen que ejecutarse necesariamente de forma periódica, esto implica que no se puede permanecer en el bucle de espera por tiempo indefinido.
3. Hay problemas cuando se quiere atender a varios periféricos. Mientras el computador espera a que un periférico esté preparado para transmitir no puede estar atendiendo a los otros.

E/S POR INTERRUPCIONES

La idea básica del mecanismo de E/S por interrupciones consiste en eliminar el bucle de espera. La UCP envía un orden de E/S al periférico y prosigue con la tarea que estaba ejecutando, en lugar de quedarse esperando a que se efectúe la operación de E/S. Cuando el periférico está preparado para intercambiar información, fuerza una interrupción en la tarea que realiza la UCP para que atienda a la operación de E/S. En ese momento la UCP realiza la transferencia de datos, de la misma manera que en el caso de E/S controlada por programa, y a continuación sigue ejecutando el programa que había interrumpido. El periférico advierte a la UCP que está preparado para la transmisión, activando una línea especial del bus de control: La línea de petición de interrupción PI. Esta forma de funcionamiento tiene alguna analogía con la ejecución de un subprograma.

La UCP no está siempre en disposición de aceptar peticiones de interrupción por parte de los periféricos. Como ejemplos de situaciones de este tipo se pueden citar las siguientes:

- Cuando la UCP no precisa de ninguna transferencia de datos con el periférico.
- Cuando la UCP está atendiendo la interrupción de un periférico muy prioritario, no debería aceptar peticiones de interrupción de periféricos poco prioritarios.

Cuando un periférico activa la señal PI, la mantiene así hasta que sabe que la UCP ha aceptado la petición. Esto significa que PI estará activa durante la ejecución del programa de servicio a la interrupción, al menos hasta que se accede al periférico en cuestión. Es esencial garantizar que en este periodo de tiempo la señal PI que está activa no origina una segunda interrupción errónea, de una interrupción simple como si fuesen múltiples peticiones, podría dar lugar a que el sistema entrase en un bucle del que no pudiese salir. Existen básicamente dos formas de conseguir esto:

- Una vez que la UCP atiende la petición de una interrupción (reconoce que $PI=1$), ignora esta línea hasta que ha completado la ejecución de la primera instrucción del programa de servicio de la interrupción. La primera instrucción de este programa es precisamente “desactivar interrupción” y así se asegura que no ocurrirá ninguna interrupción más hasta que se ejecute la instrucción inversa de “activar interrupción”. Esta instrucción será la última antes de la instrucción de “retorno” desde el programa de servicio al programa interrumpido. La UCP debe garantizar también que durante la ejecución de la instrucción de retorno no se produce ninguna interrupción.
- La UCP desactiva de forma automática el sistema de interrupciones antes de comenzar la ejecución del programa de servicio. La forma de hacerlo es poner a 1 un bit de enmascaramiento de interrupciones en el registro de estado de la UCP (lo que equivale a ejecutar la instrucción anterior de “desactivar interrupción”) antes de guardar el contexto del programa interrumpido.

Análogamente, a la vuelta del programa de servicio de la interrupción, la UCP también de forma automática activa nuevamente el sistema de interrupciones colocando un 0 en el bit de enmascaramiento de interrupciones.

Clasificación de las interrupciones

El planteamiento anterior es válido cuando existe un único periférico conectado a la línea de petición de interrupciones. El problema se complica cuando hay más de un dispositivo y cada uno de ellos puede solicitar de forma independiente la petición de una interrupción. La forma habitual de efectuar la consulta sobre las interrupciones que aún no han sido atendidas es al final de cada instrucción. La UCP examina si hay alguna petición de interrupción pendiente. Si la instrucción es de larga duración, la consulta se hace en determinados puntos de la ejecución de la instrucción. Las interrupciones se pueden estudiar atendiendo a múltiples criterios que no son mutuamente excluyentes entre sí:

Origen	a) <i>Externa</i> . Las provoca un periférico. b) <i>Interna</i> . Las provoca la UCP (p.e. dividir por 0). c) <i>Simuladas</i> . Son interrupciones software.
Nº de líneas de interrupción	a) <i>1 línea</i> . Sólo 1 línea de petición de interrupción PI. b) <i>Múltiples líneas</i> . PI_1, PI_2, \dots, PI_n .
Control de UCP sobre la interrupción	a) <i>Enmascarables</i> . La UCP puede desactivarlas. b) <i>No enmascarables</i> . La UCP no puede desactivarlas.
Identificación de la fuente de la interrupción	a) <i>Múltiples líneas</i> . PI_1, PI_2, \dots, PI_n . b) <i>Encuesta</i> . La interrupción se identifica por programa. c) <i>Vectorizadas</i> . La interrupción identifica al periférico.
Gestión de la prioridad de la interrupción	a) <i>Por software</i> . Un programa determina la prioridad. b) <i>Por hardware</i> . Un circuito determina la prioridad.
Niveles de interrupción	a) <i>Nivel único</i> . La interrupción no puede interrumpirse. b) <i>Multinivel</i> . Anidamiento de interrupciones.

Identificación de la interrupción y gestión de su prioridad

La forma más directa de identificar la fuente de una interrupción es proporcionar *múltiples líneas* de petición PI_1, PI_2, \dots, PI_n entre la UCP y los controladores de E/S. La UCP es capaz de reconocer a través de qué entrada le ha llegado la

petición de interrupción y hacer que se ejecute el correspondiente programa de servicio. Sin embargo, es poco práctico dedicar cada línea de petición de interrupción exclusivamente a un único periférico.

La situación normal es que el número de líneas de que se dispone, sea menor que número de periféricos que pueden solicitar interrupción. Es preciso reunir las peticiones de interrupción de varios periféricos en una única línea. Cuando sólo hay una línea de petición de interrupción se plantea a la UCP el problema de diferenciar cual de los periféricos conectados es el que la ha solicitado para poder atenderle de manera específica. Hay dos alternativas:

Una es la *Identificación por encuesta*: Al detectar la UCP que se ha activado la línea de petición de interrupción (es decir, $PI=1$) lo primero que hace es ir a un programa de servicio de interrupciones que interroga a cada uno de los controladores de E/S para determinar cual de ellos originó la interrupción. La encuesta se puede realizar mediante una línea de control independiente de PI. En este caso la UCP activa esa línea y coloca secuencialmente sobre el bus de direcciones, la dirección de cada uno de los controladores de E/S que tiene conectados, hasta que uno de ellos le responde de manera positiva (activando una determinada línea de control) si fue el que originó la interrupción. El programa de servicio también puede examinar secuencialmente el registro de estado RE de cada controlador de E/S hasta encontrar el que activó la línea de petición de interrupción. Una vez que se ha identificado el periférico causante de la interrupción, la UCP comienza a ejecutar un programa de servicio de interrupción específico para esa interrupción. Su desventaja es el tiempo que pierde la UCP interrogando los diferentes controladores de E/S. El método garantiza un mecanismo de gestión de prioridades cuando dos o más periféricos solicitan simultáneamente una petición de interrupción. El orden en que se comprueba si el periférico ha interrumpido o no, determina la prioridad de cada interrupción.

La segunda alternativa es la de Interrupciones vectorizadas, es una técnica más eficaz, que proporciona un mecanismo de encuesta de tipo hardware. Con la señal de petición de interrupción \overline{PI} (\overline{PI} significa que está activa en baja), el periférico envía a la UCP un vector de interrupción (*vinc*) que, de forma directa o indirecta, determina el comienzo del programa de servicio específico de esa interrupción. La secuencia de acciones que tiene lugar es la siguiente:

1. El periférico que hace la petición provoca que $\overline{PI} = 0$.
2. La UCP reconoce la petición de interrupción y activa la señal RI ($RI=1$).
3. La señal RI la recibe el periférico 1 en su entrada p_e (prioridad de entrada). Si el periférico 1 no la ha originado, propaga la señal RI a su terminal de salida p_s , poniéndola a 1 (prioridad de salida).
4. El proceso del paso 3) se va repitiendo hasta que se llega al periférico i que tiene pendiente la interrupción. Este periférico bloquea la señal de reconocimiento de interrupción, poniendo un 0 en su salida p_s , y enviando su vector de interrupción ($vinc_i$) por el bus de datos a la UCP.
5. A partir de este punto, todos los periféricos que tienen un 0 en su entrada p_e generan un 0 en su salida p_s , lo que sirve para informar al siguiente periférico que la señal de reconocimiento de interrupción ha sido ya bloqueada.

Esta técnica se conoce como *interrupciones encadenadas (daisy chain)*. Cuando hay peticiones simultáneas está implícita la prioridad de los periféricos. La máxima prioridad la posee el periférico 1 que es el que está más próximo a la UCP y la mínima el periférico n :

Controlador de interrupciones

Cuando la UCP tiene múltiples líneas de interrupción PI_1, PI_2, \dots, PI_n y se producen algunas peticiones simultáneas, es preciso dotar al sistema de algún mecanismo de selección de la interrupción más prioritaria. Cualquiera que sea la técnica empleada, siempre existirá una versión equivalente que se puede hacer por el programa de servicio de interrupciones, aunque a costa de una pérdida de velocidad en atender a la interrupción. Por este motivo se suelen emplear *soluciones hardware* que no incurren en un retraso adicional. Una de las alternativas de este último tipo consiste en emplear un *controlador de interrupciones*. El cometido básico del controlador de interrupciones PIC (*Programmable Interrupt Controller*) es ampliar el número de líneas de interrupción de la UCP y encargarse de toda la gestión del sistema de interrupciones. Las funciones que realiza el PIC son las siguientes:

- a. Identificar la fuente de la interrupción.
- b. Establecer las prioridades de cada periférico.
- c. Activar o desactivar de forma selectiva las peticiones de interrupción que recibe.
- d. Envío de información sobre la petición de la interrupción y el periférico que se debe atender.

El PIC resuelve las peticiones simultáneas de varios periféricos mediante un *codificador de prioridad*. La lógica del codificador es tal que si llegan al mismo tiempo dos o más entradas, tiene preferencia la de prioridad más alta. Otra de las funciones asignadas al PIC es la de activar o desactivar de forma selectiva las peticiones de interrupción que recibe, de manera que se autoriza que determinadas solicitudes lleguen a la UCP mientras que otras quedan enmascaradas. Hay dos formas básicas de realizar este enmascaramiento: individualmente o por nivel.

Si se realiza *individualmente* cada una de las n entradas al PIC de petición de interrupción está controlada por una puerta AND de 2 entradas, cuya segunda entrada es el bit correspondiente de un *registro de máscara de interrupciones*. Los bits del registro de máscara son accesibles por programa y la UCP escribe en este registro el valor deseado. Las entradas de interrupción con 1 en el registro de máscara están autorizadas, mientras que a las que le tengan un 0 están denegadas. El

registro de estado almacena las peticiones de interrupción que han sido autorizadas por el registro de máscara. Este registro puede ser leído por la UCP con el fin de identificar el origen de la interrupción. Las peticiones de interrupción se deben resolver por programa.

En el enmascaramiento *por nivel* las peticiones de interrupción se ordenan atendiendo a un criterio de prioridad que el codificador de prioridades se encarga de resolver. El codificador entrega a su salida un código que indica la entrada de mayor prioridad que ha solicitado una interrupción. La UCP por su parte fija un *nivel* que puede ser modificado también por programa. Todas las interrupciones que tengan un nivel superior al especificado pueden ser atendidas, mientras que las de nivel igual o inferior quedan prohibidas.

Interrupciones multinivel: Anidamiento de interrupciones

En determinadas situaciones, cuando se dispone de periféricos muy prioritarios, tener un único nivel de interrupciones esta forma de actuación supone una seria limitación, será más conveniente hacer uso de una estructura de interrupciones multinivel, en la que se pueden atender peticiones de interrupción durante la ejecución del programa de servicio de otro dispositivo. Para facilitar la realización, es conveniente asignar a la UCP un nivel de prioridad que se puede modificar por programa. El nivel de prioridad de la UCP es la prioridad del programa que se está ejecutando. La UCP acepta interrupciones sólo de aquellos dispositivos que tienen una prioridad mayor a la suya. Una petición de interrupción enmascara a las demás líneas menos prioritarias.

ACCESO DIRECTO A MEMORIA (DMA)

La E/S por interrupciones, aunque es más eficaz que la E/S controlada por programa, requiere también la intervención activa de la UCP para transferir datos entre la memoria y un periférico. En ambos casos, cualquier transferencia de datos ha de recorrer un camino que pasa a través de la UCP. Estas dos formas de E/S sufren de dos desventajas:

- La transferencia de datos está limitada por la velocidad con que la UCP puede comprobar y atender a un periférico.
- La UCP está obligada a gestionar la transferencia de E/S; hay que ejecutar una serie de instrucciones durante cada operación de E/S.

Cuando se mueven grandes cantidades de datos, se necesita una técnica más eficaz la técnica de *acceso directo a memoria* o *DMA* (Direct Memory Access).

Controlador de DMA

El DMA necesita un módulo adicional conectado al bus de sistema, el *controlador de DMA* que es capaz de hacer las funciones asignadas a la UCP y asumir el control del sistema. Cuando la UCP desea leer o escribir un bloque de datos emite una orden al controlador de DMA enviándole la siguiente información:

- Si la operación de E/S es de lectura o de escritura.
- La dirección del periférico.
- La posición de comienzo en memoria de donde hay que leer o donde hay que escribir.
- El número de palabras que se tienen que leer o escribir.

A partir de este momento la UCP continúa realizando otra tarea. La UCP ha delegado esta operación de E/S en el controlador de DMA y es este módulo quien se encargará de ella. El controlador de DMA transfiere directamente, palabra a palabra, el bloque completo de datos entre el periférico y la memoria, sin pasar por la UCP. Cuando la transferencia finaliza el DMA envía una señal de interrupción a la UCP. De esta forma la UCP únicamente participa al comienzo y al final de la transferencia.

Transferencia de datos mediante DMA

El controlador de DMA necesita tener el control del bus para poder transferir datos hacia (o desde) la memoria:

- *Por ráfagas* : Cuando el DMA toma el control del bus no lo libera hasta haber transmitido el bloque de datos pedido. Con este método se consigue la mayor velocidad de transferencia pero se tiene a la UCP inactiva durante períodos relativamente grandes. También se conoce como *parada del procesador*.
- *Por robo de ciclos*: Cuando el DMA toma el control del bus lo retiene durante un solo ciclo. Transmite una palabra y libera el bus. Es la forma más usual de transferencia y en ella el DMA roba ciclos a la UCP. El robo de ciclos reduce al máximo la velocidad de transferencia y la interferencia del controlador de DMA sobre la actividad de la UCP.
- *DMA transparente*: Es posible eliminar completamente la interferencia entre el controlador de DMA y la UCP. Se consigue si se diseña el DMA de forma que solamente se roban ciclos cuando la UCP no está utilizando el bus de sistema. Análogamente al caso de robo de ciclos no se obtiene la ventaja de una velocidad de transferencia muy elevada propia del DMA.
- *Por demanda*: El periférico es quien comienza la transferencia por DMA, pero devuelve el control a la UCP cuando no tiene más datos disponibles. Tan pronto como ese periférico tiene nuevos datos vuelve a retomar el control del bus y se continúa así hasta que acaba la transferencia completa del bloque.

- *Dato a dato*: Cada vez que el periférico solicita una transferencia por DMA se envía un único dato y se devuelve el control a la UCP. El método resulta de utilidad cuando se desea la ejecución simultánea de un programa con la recepción o transmisión de datos a velocidades moderadas (por ejemplo vía serie). La UCP no tiene que ocuparse para nada de la operación y sigue ejecutando su programa, casi con la misma velocidad, mientras que de forma simultánea se efectúa la transferencia de datos.

La UCP puede suspender el ciclo de instrucción justamente antes de que se necesite utilizar el bus. El controlador de DMA entonces transfiere una palabra y devuelve el control a la UCP, esto no es una interrupción y la UCP no guarda su contexto, ya que el controlador de DMA no altera los registros de la UCP; lo que hace la UCP es parar su actividad durante un ciclo del bus. El efecto total es que la UCP ejecuta su programa mas lentamente. Los pasos en la transferencia DMA son:

1. La UCP ejecuta dos instrucciones de E/S que cargan los registros de dirección y contador de palabras del controlador de DMA. El registro de dirección debe contener la dirección base de la zona de memoria principal que se va a utilizar en la transferencia de datos. El registro contador de palabra almacena el número de palabras que se transfieren desde (o hacia) memoria.
2. Cuando el controlador de DMA está preparado para transmitir o recibir datos, activa la línea de “petición de DMA” a la UCP. La UCP espera en el siguiente punto de ruptura del DMA, renuncia al control de los buses de datos y direcciones y activa la línea de “reconocimiento de DMA”. Las peticiones simultáneas de transferencia mediante DMA por parte de algunos controladores se resuelven utilizando las técnicas de control con prioridad del bus.
3. El controlador de DMA transfiere ahora directamente los datos a o desde la memoria principal por alguno de los métodos que se acaban de ver. Después de transferir una palabra, el registro de dirección y el registro contador de palabras del controlador se incrementa y decrementa respectivamente.
4. Si el registro contador de palabra no es 0, pero el periférico no está preparado para enviar o recibir el próximo lote de datos, el controlador de DMA devuelve el control a la UCP liberando el bus del sistema y desactivando la línea de petición de DMA. La UCP responde desactivando la línea de reconocimiento de DMA y continuando con su operación normal.
5. Si el contenido del contador de palabras es 0, el controlador de DMA renuncia al control del bus del sistema y envía una señal de interrupción a la UCP.

Configuración del DMA

El DMA se puede configurar de tres formas diferentes, todos los módulos comparten el mismo bus del sistema. El controlador de DMA, que actúa como un sustituto de la UCP, utiliza E/S controlada por programa para intercambiar datos entre la memoria y un periférico a través del controlador de DMA. Esta configuración, aunque puede ser muy económica, es claramente poco eficaz, ya que cada transferencia de una palabra consume 2 ciclos del bus igual que con la E/S controlada por programa. Una segunda forma es la de integrar las funciones de DMA y E/S. De esta forma se puede reducir substancialmente el número de ciclos de bus necesitados. Esto significa que hay un camino entre el controlador de DMA y uno o más controladores de E/S que no incluyen al bus del sistema. La lógica del DMA puede ser una parte de un controlador de E/S o puede ser un módulo independiente que controla a uno o más controladores de E/S. La tercera forma es utilizar un bus de E/S para conectar los controladores de E/S al controlador de DMA. Reduce a una el número de interfaces de E/S en el controlador de DMA y proporciona una configuración fácilmente ampliable.

En todos los casos el bus del sistema, que el controlador de DMA comparte con la UCP y la memoria, lo utiliza únicamente para intercambiar datos con la memoria. El flujo de datos entre el DMA y los controladores de E/S tienen lugar fuera del bus del sistema.

PROCESADOR DE E/S (PE/S)

Un cambio fundamental es la introducción del concepto de controlador de E/S capaz de ejecutar un programa. Los controladores de E/S se suelen denominar

1. *Canal de E/S*: Se potencia al controlador de E/S para convertirlo en un procesador con un conjunto de instrucciones especializadas en operaciones de E/S. La UCP dirige al procesador de E/S (PE/S) para que ejecute un programa de E/S que está residente en la memoria. El PE/S busca y ejecuta ese programa sin la intervención de la UCP y permite a la UCP especificar una secuencia de actividades de E/S que sólo se interrumpe cuando se ha ejecutado la secuencia completa.
2. *Procesador de E/S (PE/S)*: El controlador de E/S tiene una memoria local y se puede considerar que es un computador. Con esta arquitectura se consigue controlar un gran número de periféricos con una intervención mínima de la UCP. Tal configuración se emplea normalmente para controlar la comunicación con terminales interactivos. El PE/S se encarga de la mayor parte de las tareas que son necesarias.

Características de los PE/S

El PE/S presenta una extensión del concepto de DMA. Un PE/S tiene la capacidad de ejecutar instrucciones de E/S, lo que da un control completo sobre dicha operación. En los computadores que incluyen PE/S, la UCP no ejecuta

instrucciones de E/S, éstas se almacenan en memoria principal para ser ejecutadas por un PE/S. Así, la UCP inicia la transferencia de E/S al dar una orden al PE/S para que ejecute un programa en memoria. El programa especifica entre otras cosas las siguientes:

- a. El periférico (o periféricos) que interviene en la operación de E/S.
- b. La zona de memoria utilizada en la transferencia.
- c. Las prioridades.
- d. Que acciones hay que efectuar si se producen ciertas condiciones de error durante la transferencia.

El PE/S sigue estas instrucciones y controla la transferencia de datos. Los dos tipos de PE/S más comunes que se suelen emplear son *canal selector* y *canal multiplexor*.

El canal selector controla múltiples dispositivos de alta velocidad, en cualquier instante de tiempo está dedicado a la transferencia de datos con uno sólo de estos dispositivos. Cada dispositivo, o un pequeño número de ellos, está manejado por un controlador de E/S. El PE/S sustituye a la UCP para supervisar el funcionamiento de los controladores de E/S.

El canal multiplexor puede controlar de forma simultánea operaciones de E/S con múltiples dispositivos. Para periféricos de baja velocidad, un multiplexor orientado a la “transferencia de bytes” acepta o transmite caracteres de la forma más rápida posible a o desde los dispositivos con los que está conectado. Para dispositivos de alta velocidad, un multiplexor orientado a la “transferencia de bloques”, alterna bloques de datos de algunos dispositivos.

Gestión del sistema de E/S

Principios generales de diseño del Sistema Operativo

Los programas de aplicación de los usuarios ha de poder efectuar sus operaciones de E/S con total independencia del dispositivo. El SO debe encargarse de los problemas originados por el hecho de que estos dispositivos sean distintos y requieran manejadores diferentes. Además, la independencia del dispositivo se traduce en la designación uniforme a cada uno de ellos mediante un nombre simbólico. En Unix los dispositivos se referencian exactamente igual que si fuesen archivos.

La idea de independencia de dispositivo implica un principio de generalidad que trata de utilizar todos los dispositivos con una visión uniforme considerando la forma en que los procesos ven a los dispositivos de E/S y la forma en que el SO gestiona los dispositivos y las operaciones de E/S. Es difícil de lograr debido principalmente a la enorme variedad de dispositivos. La estrategia que se adopta es utilizar un enfoque modular y jerárquico en el diseño de la función de E/S.

Las operaciones de E/S son con frecuencia el cuello de botella de una gran parte de los sistemas, lo que implica que se debe controlar la eficiencia.

Otro aspecto importante es el manejo de errores, los errores se deben tratar lo más cerca posible del propio dispositivo físico. Si el controlador descubre un error de lectura o escritura debe tratar de corregirlo, si no puede hacerlo es el manejador del dispositivo el encargado de ello.

La forma de la transferencia puede ser asíncrona (controlada por interrupciones) o síncrona (por bloques). Los dispositivos de E/S son en su mayoría de naturaleza asíncrona.

Se debe considerar también la diferencia entre dispositivos de uso exclusivo y dispositivos que se pueden compartir. Los dispositivos de uso exclusivo presentan una serie de problemas como el del *interbloqueo*.

Estructura lógica de la función de E/S

Funciones separadas por:

- a. Complejidad.
- b. Escala de tiempos.
- c. Nivel de abstracción.

Las operaciones de E/S se estructuran por capas. Este método tiene las como ventaja que permite alcanzar los objetivos expuestos en el apartado anterior. Cada nivel realiza un subconjunto de las funciones de E/S necesitadas por el SO. Cada nivel descansa sobre el siguiente nivel inferior para efectuar funciones más básicas. Cada nivel esconde los detalles a los niveles superiores a los que proporciona servicios. Permite la descomposición del problema original en una serie de subproblemas más manejables.

Idealmente, los niveles se deben definir de manera que los cambios que se realicen en un determinado nivel no requieran modificación alguna en los otros niveles (estructura modular). Las cuatro capas en las que se estructura la función de E/S son las siguientes:

Manejador de interrupciones

Depende del dispositivo físico en cuestión. Está en contacto directo con el dispositivo físico. Cuando un proceso tiene una E/S, se inicia la operación, conmuta al estado de bloqueado y espera hasta que acabe la operación de E/S. El proceso se puede bloquear a sí mismo, por ejemplo, mediante una operación *down* en un semáforo, una operación *wait* en una variable de condición o una operación *receive* en un mensaje. El desbloqueo se puede realizar mediante una operación *up* (semáforo) o una operación *signal* (variable de condición). En una gran parte de los dispositivos de E/S las operaciones de leer y escribir comienzan y acaban por medio de interrupciones.

Manejadores de dispositivos

Depende del dispositivo físico en cuestión. Su misión es aceptar las solicitudes que hace el software del nivel superior (independiente del dispositivo) y verificar su ejecución. Tienen todo el código que depende del dispositivo. P.e. son los únicos que conocen el número de registros de un disco. Cuando se hace una solicitud de E/S en el disco, lo primero es traducirla de términos abstractos a términos concretos. El manejador del disco debe verificar el lugar donde se halla el bloque solicitado, si funciona el motor o si el brazo está en el cilindro adecuado. El manejador escribe las órdenes adecuadas en los registros del dispositivo del controlador. Dos situaciones:

- El manejador debe esperar hasta que el controlador realice algún trabajo, bloqueándose hasta que ocurre una interrupción que lo despierta.
- La operación se hace sin retraso por lo que el manejador no se tiene que bloquear.

Se deben verificar los errores después de terminar la operación. Si todo es correcto, el manejador podrá transmitir los datos al programa que los solicitó.

Software del SO independiente del dispositivo

Es independiente del dispositivo físico en cuestión. Las funciones que se le asignan son las siguientes:

- Realizar las tareas de E/S comunes a todos los dispositivos, proporcionando un interfaz uniforme del software a nivel de usuario.
- Asociar los nombres simbólicos de los dispositivos con el nombre adecuado.
- Proporcionar esquemas de protección a los dispositivos. En sistemas monousuario, como MS-DOS, no existe protección, pero en sistemas multiusuario o multitarea sí. Por ejemplo, en Unix se asocian tres bits (*rwx*, *lectura escritura ejecución*) para la protección de los dispositivos.
- Proporcionar un tamaño uniforme de bloque a los niveles superiores de software.
- Utilización de buffers, de manera que aunque la E/S se haga por bloques, los procesos de usuario puedan leer y escribir unidades arbitrarias.
- Asignación de espacio en los dispositivos por bloques.
- Asignación y liberación de los dispositivos de uso exclusivo, como las impresoras.
- Informar a los niveles superiores de la jerarquía de los errores y del tratamiento dado a los mismos (esta función la gestiona el manejador).

Software a nivel de usuario

Es independiente del dispositivo físico en cuestión. Una parte del software de E/S no está dentro del SO, sino en librerías accesibles a los programas de usuario. El sistema de *spooling* crea un acceso especial denominado *demonio* y un directorio especial que es el *directorio de spooling*. Cuando un proceso quiere imprimir un archivo, efectúa las dos acciones siguientes:

1. Genera el archivo que se desea imprimir.
2. Coloca el archivo en el directorio de spooling.

Es el demonio, el único proceso autorizado para usar la impresora, el que va imprimiendo los archivos colocados en el directorio de spooling. Esta técnica no sólo se usa para las impresoras, sino que también la utilizan, por ejemplo, los servicios de red para la transferencia de archivos.

Buffers de E/S

Espacios de memoria principal que se reservan para el almacenamiento intermedio de datos procedentes o con destino a los periféricos. Los dos esquemas más usuales de reserva de zonas que incorporan los Sistemas Operativos para mejorar su rendimiento son el buffer simple y el buffer doble.

Buffer simple

Transferencias de entrada:

1. La transferencia de un bloque de la entrada se hace desde el dispositivo al buffer que el SO le reserva, al proceso que hace la petición, en la memoria principal.
2. Cuando finaliza la transferencia anterior, el proceso mueve el bloque desde el buffer al espacio del proceso del usuario.
3. Inmediatamente se solicita otro bloque.

Las transferencias de salida siguen el camino inverso:

1. El bloque que se desea transferir se copia desde el espacio del proceso del usuario al buffer que el SO le reserva en la memoria principal.
2. El bloque se escribe desde el buffer en el controlador de E/S.
- 3.

Representa una mejora de velocidad: el proceso puede estar procesando un bloque mientras se lee otro. El SO debe mantener un registro de las asignaciones que se hacen entre los buffers del sistema con los procesos de los usuarios. El buffer simple también se puede utilizar en el caso de dispositivos orientados a caracteres de dos formas diferentes:

1. Procesamiento de toda la línea de caracteres. En una operación de entrada el proceso del usuario se suspende esperando la llegada de la línea completa. En la operación de salida, el proceso puede colocar toda la línea en el buffer y continuar su procesamiento.
2. Procesamiento carácter a carácter. Se emplea en terminales utilizados en modo "formulario" donde la pulsación de cada tecla es significativa.

Buffer doble

Un proceso transfiere datos a (o desde) uno de los buffers mientras el SO vacía (o llena) el otro buffer. Para dispositivos de entrada orientados a caracteres existen métodos alternativos de operación:

1. Procesamiento de toda la línea de caracteres. El proceso del usuario no necesita suspenderse para operaciones de entrada o de salida a menos que vaya por delante de los dos buffers.
2. Procesamiento carácter a carácter. El doble buffer no ofrece ninguna ventaja frente a uno simple de longitud doble.

Comparación de los tiempos de ejecución

Sin buffer

$$T = t_B + t_C$$

Con buffer simple

$$T = \max(t_B, t_C) + t_M$$

En la mayoría de casos es substancialmente menor que la anterior.

Con buffer doble

$$T = \max(t_B, t_C)$$

T = tiempo de ejecución por bloque.

t_B = tiempo requerido para transferir un bloque.

t_C = tiempo de cálculo que media entre dos peticiones de transferencia.

t_M = tiempo requerido para transferir el bloque desde el buffer a la memoria del proceso del usuario.

Es posible, por lo tanto, mantener el dispositivo orientado a bloques trabajando a su máxima velocidad si $t_C < t_B$. Por otra parte si $t_C > t_B$ el empleo de un buffer doble asegura que el proceso no tendrá que esperar por la operación de E/S. En cualquiera de los dos casos se obtiene una mejora si se le compara con el caso del buffer simple.

Discos Magnéticos

Estructura física

Formado por una película de óxido magnético que recubre un disco. Estos platos pueden ser extraíbles (giran a menos velocidad) o fijos (giran a más de 3000 r.p.m.). Una unidad también puede disponer de varios platos. El número varía entre 1 y 20. Los datos se leen y escriben mediante cabezas de lectura/escritura montadas de forma que contacten con la parte del disco que contiene los datos. Cada disco tiene dos superficies (o caras) por lo que hay dos cabezas de lectura/escritura por cada disco. Los datos se almacenan en las superficies magnéticas del disco en forma de círculos concéntricos llamados pistas. Se denomina cilindro al conjunto de pistas de todas las superficies que están a la misma distancia del eje de los diferentes discos. Las pistas se dividen en sectores y cada sector contiene varios centenares de bytes. La densidad de grabación en las pistas internas será superior que en las externas ya que los sectores interiores son más pequeños y en cambio el número de bytes por sector y el número de sectores por pista es constante. Dependiendo del número de cabezas, los discos pueden ser de

- **Cabeza fija**. Tienen, generalmente, una cabeza de lectura/escritura por cada pista. Así, para leer una pista sólo se debe activar la cabeza correspondiente.

$$\text{tiempo de acceso } (t_a) = \text{tiempo de latencia rotacional } (t_r)$$

El t_r , en promedio, es igual a la mitad de la velocidad de revolución del disco.

- **Cabeza móvil**. Tiene una o unas pocas cabezas de lectura/escritura por cada superficie. Esto quiere decir que para acceder a un sector, la cabeza debe desplazarse para situarse sobre él antes de activarla.

$\text{tiempo de acceso } (t_a) = \text{tiempo de latencia rotacional } (t_r) + \text{tiempo de posicionamiento de la cabeza o tiempo de búsqueda } (t_b)$

Una vez alcanzado el sector la velocidad de transferencia de datos de los discos actuales suele ser del orden de 5 Mbytes por segundo. A causa de estas velocidades lo normal es usar técnicas de acceso por DMA para la transferencia de datos entre los discos y la memoria principal.

El tiempo que se precisa para que los datos se transfieran desde el disco a la memoria principal del computador se compone de los tres tiempos siguientes:

Tiempo de búsqueda (t_b): Tiempo necesario para que las cabezas se desplacen al cilindro adecuado. Consta de dos componentes clave el tiempo de arranque inicial (t_i) y el tiempo que se tarda en recorrer todos los cilindros que hay entre la pista inicial y la final. Se suele aproximar con la fórmula siguiente

$$t_b = m \times n + t_i$$

Retardo rotacional (t_r): Tiempo medio que tarda el sector en estar debajo de la cabeza de lectura/escritura. Si f es la velocidad de rotación en revoluciones por segundo,

$$t_r = \frac{1}{2 \times f}$$

Tiempo de transferencia (t_r): Tiempo que se tarda en transferir los datos. Depende del tamaño del sector y es el menor de los tres tiempos. Si b es el número de bytes que se desean transferir y P el número de bytes que hay en una pista

$$t_i = \frac{b}{P \times f}$$

En algunos supercomputadores, con el fin de mejorar el rendimiento de la transferencia de datos entre el disco y la memoria principal, se suele emplear una técnica conocida como sensor de posición rotacional: Cuando se emite una orden de “búsqueda de sector” se libera el canal para manejar las operaciones de E/S. Al finalizar esta orden, el dispositivo determina en que momento el sector pasará por encima de la cabeza y cuando se va aproximando intenta establecer de nuevo la comunicación. En el caso de que la Unidad de Control del computador o el canal estén ocupados atendiendo a otra operación de E/S fallará el intento de reconexión y el dispositivo deberá girar una vuelta completa antes de volver a intentarlo otra vez.

Controlador del disco

Las señales típicas entre el controlador del disco y la unidad propiamente dicha son:

- a. *Líneas de selección de unidad.* Seleccionan la unidad de disco a la que se accederá.
- b. *Líneas de selección de cabeza.* Activan la cabeza de lectura/escritura adecuada.
- c. *Línea de dirección.* Indica si la cabeza se desplazará hacia adentro o hacia afuera.
- d. *Línea de paso.* Proporciona una secuencia de impulsos.
- e. *Línea de leer.* Indica que se desea leer.
- f. *Línea de escribir.* Indica que se desea escribir.
- g. *Salida de datos.* Transporta los bits leídos del disco.
- h. *Entrada de datos.* Transporta los bits que se escribirán en el disco.
- i. *Reinicializar.*
- j. *Indicación de Fallo.*

El control de errores es necesario para detectar los errores transitorios provocados por ruidos e interferencias electromagnéticas y para manejar los defectos del medio de almacenamiento. Cada vez que se escribe un dato en el disco, el controlador calcula los bits de verificación y los añade a la información que se transmite. Como los errores de disco ocurren en secuencias y producen errores en cadenas de bits sucesivos, los controladores suelen utilizar mecanismos de detección capaces de tratarlos, como los códigos de redundancia cíclica (CRC). Dependiendo de la causa, un error puede ser transitorio o permanente. Si es transitorio se corrige leyendo varias veces el sector correspondiente. El número de intentos depende de un parámetro llamado *contador de reintentos*. En el caso de que el error sea permanente si los reintentos de corrección fracasan se marca como defectuoso, de forma que se impide su utilización.

Otros errores, denominados *errores de búsqueda*, pueden deberse a problemas mecánicos de las cabezas. Un ejemplo de esta clase puede ser que el brazo de la cabeza de lectura/escritura no se posiciona correctamente. Algunos controladores corrigen automáticamente los errores de búsqueda, pero otros activan un bit de error y pasan el resto del problema al manejador, que se encarga del error enviando una orden de recalibración.

El controlador debe disponer de un buffer interno para el almacenamiento temporal de los datos. Las razón es que una vez que se ha iniciado una operación de E/S desde el disco, los datos se transfieren a velocidad constante, independientemente de que el controlador esté preparado o no para recibirlos. Si el controlador pretende transferirlos directamente a la memoria es preciso que utilice el bus del sistema, que sin embargo podría estar ocupado por otro dispositivo. La utilización de un buffer interno hace innecesario el bus del sistema hasta que comienza el acceso por DMA (lo que facilita el diseño del controlador ya que la transferencia a memoria mediante DMA no impone limitaciones de tiempo) y tiene consecuencias importantes en la gestión del sistema de E/S. Cuando el controlador o la propia UCP están transfiriendo los datos a la memoria (operación de salida), el siguiente sector estará pasando por debajo de las cabezas de lectura de manera que la información estará llegando al controlador (operación de entrada). Si éste tiene un diseño sencillo no permite realizar de forma simultánea la entrada y la salida de datos por lo que durante la transferencia a memoria no hay posibilidad de leer el siguiente sector.

Por lo tanto, este tipo de controladores no es capaz de leer bloques contiguos de información (que están en sectores contiguos). Así, la lectura de una pista completa requiere al menos dos rotaciones, una para los sectores pares y otra para los impares. Esta técnica de saltar bloques, puede dar un margen de tiempo suficiente para que el controlador transmita los datos a la memoria, se llama *entrelazado*. Por esta razón, cuando se formatean discos, los sectores se numeran teniendo en cuenta el *factor de entrelazado*.

Las funciones que se tiene que hacer en un disco se pueden describir mediante una rutina de disco, que permita la lectura y escritura de sectores especificados por medio de direcciones físicas de la forma siguiente:

<número de cilindro, número de cabeza, número de sector>

Las órdenes elementales al nivel de la rutina son *buscar*, *leer* y *escribir*. Como las transferencias de datos entre el disco y la memoria se hace mediante DMA, la rutina de disco no devuelve datos explícitos a sus invocadores, excepto la información de resultado de la operación.

Planificación del disco

El único componente del tiempo de lectura/escritura que se puede optimizar desde el programa gestor del disco es el tiempo de búsqueda del cilindro, ya que los demás componentes dependen de las características del propio disco. Cuando un proceso requiere una operación de E/S del disco, envía la correspondiente llamada al SO especificándole la siguiente información:

- a. Tipo de operación: de entrada o de salida.
- b. Dirección en el disco (unidad, cilindro, cabeza, sector)
- c. Dirección en memoria.
- d. Cantidad de información que se va a transferir.

Si está disponible, es posible atender de forma inmediata la petición. En caso contrario será necesario poner en una cola la petición. En los sistemas multiprogramación, la mayoría de las veces la cola del disco no estará vacía, de manera que al acabar una petición habrá que elegir una de las que están pendientes de ser servidas.

Planificación FCFS: primero en llegar, primero en ser servido

Las solicitudes se almacenan en una memoria tipo FIFO.

Planificación SSTF: primero la de menor tiempo de posicionamiento

Consiste en atender la petición que requiere el menor movimiento de la cabeza de lectura/escritura desde su posición actual. De esta forma se elige la opción que incurre en el menor tiempo de búsqueda. Esto, sin embargo, no garantiza que el tiempo medio de búsqueda a lo largo de un número de movimientos sea mínimo. Como la cabeza se puede mover en las dos direcciones es posible emplear un algoritmo de “tirar la moneda” para resolver los casos de distancias iguales. Un problema potencial que se puede presentar con este algoritmo es el *bloqueo indefinido* (cierre) de algunas peticiones: las peticiones muy alejadas de la pista actual puede que nunca sean atendidas si van llegando peticiones de pistas cercanas a la actual.

Planificación SCAN

Evita el bloqueo indefinido que se puede producir con la planificación SSTF. La estrategia es ir recorriendo las pistas en una dirección y satisfaciendo todas las peticiones que se encuentran en el camino, hasta que alcanza la última pista o no hay más solicitudes en esa dirección. En ese punto se invierte el sentido del recorrido y la búsqueda prosigue de la misma forma. También se le conoce como *algoritmo del ascensor*. Proporciona ventajas a los procesos cuyas peticiones son a pistas que están localizadas en los cilindros más internos y externos del disco.

Planificación C-SCAN

Esta estrategia restringe el rastreo a una única dirección. Así, cuando se ha visitado la última pista en una dirección, la cabeza vuelve al extremo opuesto del disco y comienza otra vez la exploración. De esta forma se consigue reducir el retardo máximo que experimentan las nuevas peticiones.

Planificación LOOK y C-LOOK

Como SCAN y C-SCAN pero sólo mueven la cabeza hasta la última petición en cada dirección.

7. MULTIPROCESADORES Y SISTEMAS DISTRIBUIDOS

Esquema de clasificación para los sistemas con varios procesadores (Flynn) basado en cómo la máquina relaciona sus instrucciones con los datos que tiene que procesar:

SISD (Single Instruction Single Data). Un solo flujo de instrucciones, un solo flujo de datos, computadores con un único procesador.

SIMD (Single Instruction Multiple Data). Un solo flujo de instrucciones, varios flujos de datos, procesadores vectoriales y los computadores en array

MISD (Multiple Instruction Single Data). Flujo de varias instrucciones, un solo flujo de datos. Ningún computador conocido se ajusta a este tipo.

MIMD (Multiple Instruction Multiple Data). Flujo de varias instrucciones, flujo de varios datos. Corresponde a un grupo de computadores independientes cada uno con su propio contador de programa, programa y datos.

Dentro de la categoría MIMD se pueden considerar otro tipo de división según la conexión entre los procesadores:

Sistemas fuertemente acoplados. Los procesadores comparten la memoria y el reloj. Son *sistemas multiprocesadores* y la comunicación generalmente se hace a través de la memoria compartida.

Sistemas débilmente acoplados. Los procesadores no comparten la memoria ni el reloj y cada uno tiene su propia memoria local. Se comunican mediante redes locales o de área extensa, formadas mediante líneas de fibra óptica de alta velocidad o líneas telefónicas. Los *sistemas distribuidos* pertenecen a esta clase.

Un sistema distribuido proporciona a los usuarios acceso a los distintos recursos que ofrece el sistema. El acceso a estos recursos lo controla el SO. Dos esquemas básicos son:

Sistemas operativos de red. Los usuarios están enterados de la multiplicidad de máquinas y para acceder a sus recursos necesitan conectarse al computador remoto apropiado, se habla de sistemas débilmente acoplados tanto en software como en hardware.

Sistemas operativos distribuidos. Los usuarios no necesitan saber que existe una multiplicidad de máquinas y pueden acceder a los recursos remotos de la misma forma que lo hacen a los recursos locales. Son sistemas de hardware débilmente acoplado, pero de software fuertemente acoplado.

Ventajas e inconvenientes de los sistemas distribuidos

Las ventajas frente a los sistemas centralizados son:

- Ahorro económico al mejorar la relación precio/rendimiento.
- Mayor potencia de cálculo.
- Mejor adaptación a aplicaciones inherentemente distribuidas.
- Mayor fiabilidad: el fallo de un procesador no deja fuera de servicio a todo el sistema.
- Posibilidad de crecer según las necesidades.

Las ventajas frente a los computadores personales aislados son:

- Mayor flexibilidad, al permitir distribuir la carga de trabajo.
- Compartición de datos y programas.
- Compartición de periféricos, unidades de almacenamiento o impresoras.

Inconvenientes

- Existe poco software para sistemas distribuidos.
- Posibles problemas de la red: saturación o pérdidas de información en la transmisión.
- Problemas de seguridad.

Hardware de los multiprocesadores

Los multiprocesadores se pueden aplicar para aumentar la productividad del sistema, al poder ejecutarse diferentes procesos de usuario sobre algunos procesadores en paralelo. También se pueden aplicar para ganar velocidad en la aplicación, al ejecutarse en paralelo algunas partes de la aplicación. Las formas de conexión para formar un sistema multiprocesador son:

- Multiprocesadores con bus compartido.
- Sistemas conectados mediante barras cruzadas.
- Sistemas conectados mediante una red de conmutación multietapa.
- Conexiones hipercubo.

La importancia de la forma de conexión radica en la influencia dominante que tiene sobre el ancho de banda y la saturación de la comunicación en el sistema.

Multiprocesadores con bus compartido

Se usa un bus compartido para conectar los procesadores y la memoria. Para que un procesador lea una palabra de la memoria, debe colocar la dirección de la palabra deseada en las líneas de dirección del bus y activar la señal de lectura en la línea de control adecuada. La memoria responde situando el contenido de la palabra en las líneas de datos. Este esquema es coherente, pero tiene el problema de que el bus o la memoria compartida se pueden saturar, con lo que el rendimiento disminuye de forma drástica. Una posible solución es disponer de una memoria caché, bien sea asociándola con la memoria, o asociándola a cada procesador. En este último caso todas las solicitudes de memoria pasan por la cache, en caso de fallo se usa el bus.

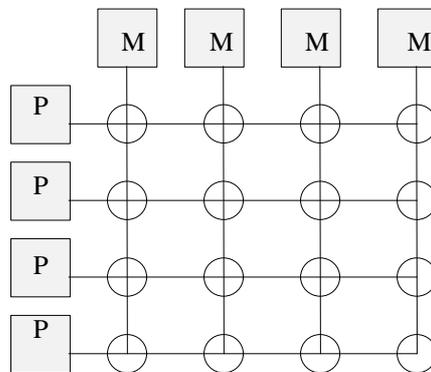
El problema es la coherencia entre las diferentes cachés. Para ello se utilizan protocolos especializados y hardware adicional, como los “espías” de caché o monitores de caché, que observan el bus y cuando se hace una petición de escritura en una dirección presente en la caché, se actualiza ésta con el nuevo valor. Pero esto hace que aumente el tráfico en el bus y disminuya el beneficio proporcionado por la utilización de la caché.

Además de las cachés, se pueden emplear otros procedimientos que disminuyan la saturación del bus. Por ejemplo, utilizar un bus de más anchura o soportar protocolos diferentes para las peticiones y las respuestas.

La escalabilidad de este sistema está limitada por el bus y la memoria compartida, en implementaciones comerciales se soportan del orden de 10 procesadores.

Sistemas conectados mediante barras cruzadas

Es una solución para construir sistemas multiprocesadores que soporten un número elevado de procesadores. Se dispone de n memorias y n procesadores conectados mediante conmutadores.



La conmutación en los cruces es la única fuente de retardo entre los procesadores y las memorias. La contención en el acceso a la memoria sólo ocurre cuando dos procesadores intentan acceder a la misma memoria al mismo tiempo. Esto aparece con las variables compartidas, como los semáforos. Este esquema permite un elevado grado de paralelismo entre tareas no relacionadas, pero se presentan problemas de contención cuando hay procesos con memoria compartida. Para n procesadores y n memorias se necesitan n^2 conmutadores. Este crecimiento cuadrático hace que el coste de estos sistemas sea elevado y por lo tanto se limite la escalabilidad del sistema.

Sistemas conectados mediante una red de conmutación multietapa

Son una solución al elevado coste de los sistemas basados en barras cruzadas, una solución típica es la *red Omega* que esta está basada en conmutadores 2×2 , cada uno con dos entradas y dos salidas; cada conmutador puede dirigir cualquiera de las entradas a cualquiera de las salidas y, con la adecuada disposición de los conmutadores, todos los procesadores pueden acceder a todas las memorias. Un conmutador multietapa puede conectar simultáneamente todas las entradas a todas las salidas siempre y cuando dos procesadores intenten un acceso simultáneo al mismo módulo de memoria, en cuyo caso aparecerá contención en el acceso.

Un problema que presenta es el *punto caliente*, se bloquean ciertas rutas de comunicación cuando algunos módulos de memoria se referencian con mucha frecuencia. Otro problema es el retraso en la conexión. Para n procesadores y n memorias se necesitan $\log_2 n$ etapas y $n/2$ conmutadores. Los conmutadores han de ser muy rápidos para que no se desperdicie tiempo del procesador a la espera de datos.

Conexiones hipercubo

Un hipercubo es un cubo n -dimensional. Son estructuras recursivas de forma que los mayores contienen hipercubos de dimensiones menores. En un hipercubo n -dimensional cada procesador tiene n conexiones con otros procesadores y la máxima distancia entre dos nodos cualesquiera (número de enlaces físicos entre nodos) es también n . El número total de nodos es 2^n .

Ya que la complejidad de los hipercubos crece logarítmicamente con el número de nodos, estos son una buena base para realizar sistemas escalables. Además la comunicación entre nodos adyacentes es directa y el retardo entre nodos está acotado

por la dimensión. Existen en el mercado hipercubos de dimensión 10 (1024 procesadores) y hasta de dimensión 14 (16384 procesadores).

Sistemas operativos para multiprocesadores

Clases de sistemas operativos para multiprocesadores

Sistemas con supervisores separados

Cada nodo tiene un SO que gestiona el procesador local, la memoria y los dispositivos de E/S. Cada supervisor gestiona cada procesador como un sistema independiente añadiendo algunos servicios y estructuras de datos para soportar el diseño de un hardware para multiprocesadores. No soportan bien el paralelismo en las aplicaciones y no está muy optimizado el reparto de la carga entre los procesadores. Los hipercubos son un ejemplo de aplicación de estos sistemas, debido a su diseño regular y repetitivo de bloques idénticos. Cada núcleo dispone de sus propios servicios de gestión de procesos, de memoria local y de paso de mensajes. El paralelismo se consigue dividiendo las tareas en subtareas. Las funciones a nivel de sistema completo, como la asignación de procesadores y la planificación global se hace en un supervisor del sistema que se implementa según alguna de las otras dos clases (maestro/esclavos o simétrico).

Sistemas maestro/esclavos

Un procesador (el maestro) se dedica a ejecutar el SO, planificando las tareas y controlando a los otros procesadores (los esclavos). También controla las estructuras de datos almacenándolas en su memoria. La mayor parte de los servicios del SO los proporciona el maestro. Los esclavos sólo pueden procesar consultas locales sencillas. Esta disposición es adecuada para el paralelismo dentro de las tareas, asignándoles varios procesadores, pero para el SO no es posible el paralelismo, ya que sólo se ejecuta en un procesador. Son fáciles de desarrollar, pero su escalabilidad está limitada por tener un solo procesador maestro.

Sistemas simétricos

Se consideran todos los procesadores funcionalmente idénticos, pudiendo realizar cualquier tarea. El SO se considera también simétrico puesto que cualquier procesador puede ejecutarlo, incluso se puede ejecutar de forma paralela en varios procesadores. La forma más sencilla de organizar un sistema simétrico es mediante un *maestro flotante*, donde el procesador que temporalmente esté ejecutando al SO actúa como maestro y planifica las tareas para el resto de los procesadores. La ventaja de un sistema de maestro flotante es que es fácil de implementar a partir de sistemas monoprocesadores, como Unix.

El paralelismo entre las diferentes tareas se consigue fácilmente manteniendo una cola de tareas y asignado el primer proceso al primer procesador disponible. El paralelismo en la ejecución del SO se puede conseguir si se identifican los procesos del sistema que pueden acceder concurrentemente a las estructuras de datos compartidas, teniendo en cuenta que el control de concurrencia con mecanismos propios de sistemas monoprocesadores fallan en un entorno de multiprocesadores (p.e., la habilitación y deshabilitación de interrupciones o el aumento temporal de la prioridad de un proceso). Estas partes del sistema se tienen que rehacer para que funcionen correctamente cuando se ejecutan en paralelo.

Gestión de los recursos

Un SO multiprocesador tiene que gestionar los procesadores, la memoria y los dispositivos de E/S. La planificación de los procesadores consiste en asignar los procesadores a los procesos, según los objetivos de diseño del sistema, y asegurar el uso eficiente de los procesadores asignados a una tarea. La productividad del sistema es función de cómo se asignen los procesadores, mientras que la velocidad depende principalmente del uso eficiente de los procesadores asignados. Productividad y velocidad pueden estar contrapuestas: mayor productividad significa más aplicaciones en uso; mayor velocidad significa mayor número de procesadores en un aplicación. El sistema operativo debe proveer mecanismos que fomenten el paralelismo.

La gestión de memoria depende en gran medida del hardware y del sistema de conexión de los procesadores. Si están débilmente acoplados la memoria se gestiona de forma independiente para cada procesador, en sistemas de memoria compartida el sistema operativo debe proporcionar un modelo flexible de la memoria, que permita un acceso seguro y eficiente a las estructuras de datos compartidas.

Los dispositivos de E/S no han requerido, hasta ahora, un tratamiento especial ya que los sistemas multiprocesadores se dedican fundamentalmente a aplicaciones que son intensivas en cálculo sin demasiadas operaciones de E/S.

Redes de computadores

Hay básicamente dos tipos de redes, las *redes locales* (LAN, *Local Area Networks*) que abarcan una extensión geográfica pequeña. Tienden a tener anchos de banda elevados y retardos de comunicación bajos, y las *redes de área extensa* (WAN, *Wide Area Networks*) que abarcan una extensión geográfica grande. Tienden a tener anchos de banda bajos y retardos de comunicación elevados.

Topología de las redes

Red de conexión total

Cada instalación está enlazada directamente con todas las demás. Tiene el inconveniente de que el coste básico de esta configuración es muy elevado, el número de conexiones crece con el cuadrado del número de instalaciones. Su ventaja es la rapidez en la comunicación y la fiabilidad del sistema (se deben romper muchas conexiones para que una parte del sistema quede incomunicada).

Red de conexión parcial

Se suprimen algunos enlaces de la red de conexión total para rebajar el coste de la instalación. Su inconveniente es que aumenta el retardo promedio en las comunicaciones y disminuye la fiabilidad, sin embargo el coste es más reducido que la de conexión total.

Red jerárquica

Tiene estructura de árbol. Alternativa muy habitual en redes corporativas. Sus inconvenientes son que aumenta el retardo promedio en las comunicaciones alejadas en parentesco y disminuye mucho la fiabilidad pues la rotura de un enlace deja la red dividida en dos subredes disjuntas. El coste es más reducido que la de conexión parcial.

Red en estrella

Tiene un coste lineal, es decir, proporcional al número de instalaciones y las comunicaciones entre sistemas requieren como máximo dos enlaces. Su principal inconveniente es que el sistema central al soportar todo el peso de las operaciones se puede saturar. Por eso en muchas ocasiones el sistema central se dedica en exclusivo a la gestión del paso de mensajes.

Red en anillo

Cada instalación está conectada a otras dos, sus vecinos. La red puede ser unidireccional (la información sólo puede viajar en una dirección) o bidireccional (la información viaja en las dos direcciones). Tiene un coste lineal con el inconveniente de un tiempo de comunicación alto.

Canal multiacceso (Bus)

Sólo existe un enlace y todas las instalaciones se conectan a él. Tiene un coste lineal, pero una rotura hace que la red quede partida en dos subredes disjuntas.

Protocolos de comunicación y el modelo de referencia OSI

El software de comunicación, debido a su complejidad, se suele organizar por capas. Cada capa proporciona una serie de servicios a los niveles superiores, la comunicación entre capas se hace mediante un interfaz bien definido. La Organización de Normas Internacionales (ISO) propuso un modelo de arquitectura de red estructurado en siete capas; es el modelo OSI (*Open Systems Interconnection*):

- *Nivel físico* . Es el responsable de la transmisión física de un flujo de bits a través de la línea de comunicación. Se encarga de los circuitos de comunicación y de sus interfaces físicas y de los procedimientos con el medio de transmisión físico.
- *Nivel de enlace de datos* . Transfiere datos entre sistemas de conexión directa y detecta errores dando a las capas superiores la sensación de un medio libre de errores.
- *Nivel de red* . Proporciona conexiones y encamina los paquetes por la red de comunicaciones.
- *Nivel de transporte* . Tiene la responsabilidad del acceso a bajo nivel a la red y transferencia de mensajes entre clientes, incluyendo la división de los mensajes en paquetes, en control de flujo y la generación de direcciones físicas.
- *Nivel de sesión* . Implementa sesiones o protocolos de comunicación entre procesos. También está encargado de conexiones remotas y de la transferencia de archivos y correo.
- *Nivel de presentación* . Resuelve las diferencias entre los formatos en las diversas instalaciones en la red, incluyendo la conversión entre caracteres y su codificación y decodificación.
- *Nivel de aplicación* . Realiza la interacción directa con los usuarios y proporciona los protocolos generalmente requeridos por los procesos de aplicación: conexión remota, transferencia de archivos y correo electrónico.

Los sistemas construidos según los protocolos OSI se llaman formalmente *sistemas abiertos*. Estos sistemas que ofrecen un interfaz en el último nivel para los programas de aplicaciones de los usuarios, se denominan *sistemas finales*. Cuando sólo contienen los tres primeros niveles, utilizados para conectar entre sí sistemas finales, se denominan *sistemas intermedios*. Los sistemas intermedios generalmente se usan en redes complejas que conectan entre sí a sistemas abiertos. Los sistemas intermedios pueden tener más de dos interfaces de conectividad, incluyendo la conexión con redes que no se ajustan al modelo de referencia OSI.

Los niveles inferiores (1 y 2) se suelen implementar en circuitos integrados. Las capas intermedias (3 y 4) tienden a residir en procesadores especializados. Los niveles superiores (5, 6 y 7) se realizan en software en el sistema final. El interfaz con los sistemas no OSI se hace en los niveles 7, 4 ó 3. Si se realiza en el nivel 7 se denomina *puerta de aplicación*, si se realiza en el 4 se llama *relevo de transporte* y si se efectúa en el nivel 3 se conoce como *encaminador*.

Redes locales (LAN)

Surgieron a principios de los años setenta como sustituto de los grandes sistemas de cálculo centralizados: era más económico tener varios computadores pequeños, que un solo sistema de gran tamaño. Una LAN se distingue por:

- El método de control de acceso a los medios.
- El método de transmisión (banda ancha o banda base).
- El medio de transmisión (par trenzado, cable coaxial o fibra óptica).

Las configuraciones más habituales son las redes de canal multiacceso, anillo y estrella, con velocidad de transmisión entre 1 Mbyte y 1 Gbyte por segundo. Las redes locales se suelen definir como una arquitectura de niveles según una norma IEEE, similar al modelo de referencia OSI y compatible con él, pero no idéntico. La principal diferencia entre los dos modelos estriba en la división del nivel 2 de OSI (enlace de datos) en dos subniveles:

- Subnivel de control de acceso a los medios.
- Subnivel de control de enlace lógico.

Un tipo de red muy extendido es el que emplea un esquema Ethernet, generalmente sobre un cable coaxial y en el que se conectan diferentes tipos de computadores, dispositivos periféricos y pasarelas de acceso a otras redes. Como no tiene controlador central su ampliación es sencilla.

Redes de área extensa (WAN)

Surgieron a finales de los años setenta como un proyecto de investigación académico para ofrecer una comunicación eficiente entre instalaciones, permitiendo compartir en hardware y el software entre usuarios. La primera red que se diseñó y desarrolló fue Arpanet (1968) que dará lugar a Internet. Debido a su extensión, estas redes son más lentas que las LAN y sus velocidades típicas de transmisión están entre los 1200 bytes y varios Mbytes por segundo.

Estas redes poseen los llamados *procesadores de comunicación* que se hallan conectados mediante comunicaciones físicas, enlaces por microondas o canales de satélites. Los procesadores de comunicación definen el interfaz por medio del cual se comunican las instalaciones de la red y al mismo tiempo se encargan de transferir información entre las instalaciones.

Una de las redes de área extensa más populares es Internet donde los computadores conectados son de todo tipo, que generalmente se encuentran en redes locales, que a su vez están conectadas a Internet mediante redes regionales. Las redes regionales están enlazadas por encaminadores (*routers*) para formar una red de alcance mundial. Los encaminadores conectan las redes en el nivel de red ISO y controlan las rutas que sigue cada paquete por la red. El encaminamiento puede ser dinámico o estático, el encaminamiento *dinámico* aumenta la eficacia de las comunicaciones, mientras que el *estático* reduce los riesgos de seguridad. La mayoría de los computadores (sobre todo los servidores) se direccionan lógicamente mediante un nombre compuesto de varias partes. La asignación del nombre se basa en un sistema de nombres de dominios, que leídos de izquierda a derecha, van de la parte más específica a la más general. Los protocolos de Internet están divididos en capas, pero no se corresponden exactamente con los de OSI. La arquitectura de Internet es:

- Nivel de *acceso a la subred*. Se encuentran los protocolos necesarios para interactuar con la red a la que físicamente está conectada.
- Nivel Internet. Encargado de la interconexión de las redes. El protocolo más importante en este nivel es el IP (*Internet Protocol*), cuyo objetivo es conseguir la interconectividad en redes heterogéneas.
- Nivel de *control de transmisión*. En este nivel están los protocolos TCP (*Transmission Control Protocol*) y UDP (*User Datagram Protocol*). Los dos están basados en IP, pero la diferencia es que TCP es un servicio de transferencia de datos orientado a conexión, mientras UDP no está orientado a conexión. El más utilizado es TCP, de ahí que muchas veces se hable de TCP/IP como el conjunto de protocolos de Internet.
- No existen los niveles orientados a las aplicaciones. Los tienen que asumir las aplicaciones. Sin embargo, existen algunos protocolos estándar para algunas aplicaciones muy comunes. Por ejemplo, están los protocolos FTP para transferencia de archivos, Telnet para el acceso interactivo, y SMTP para el correo electrónico.

Tipos de sistemas operativos para sistemas distribuidos

Los sistemas de software débilmente acoplados son los sistemas operativos de red. En estos los usuarios son conscientes de la multiplicidad de máquinas y el acceso a los recursos de otros computadores se hace por petición del usuario, conectándose al computador remoto. Los sistemas operativos realmente distribuidos corresponden a sistemas de software fuertemente acoplados. En este caso los usuarios no necesitan saber que existen múltiples máquinas y acceden a los recursos remotos de la misma forma que lo hacen para los recursos locales.

Sistemas operativos de red

Su función principal es ofrecer un mecanismo para transferir archivos o poder dar órdenes de ejecución de programas desde una máquina a otra. El computador del usuario funciona como un terminal del computador remoto. Los órdenes que se ejecuten lo harán en la máquina remota. También se pueden transferir archivos de una máquina a otra. Un

protocolo típico, usado en Internet, es el *protocolo de transferencia de archivos* (FTP): El usuario debe invocar al programa FTP, el cual lo solicitará el nombre del computador a partir del cual se realizará la transferencia y la información de acceso para verificar que el usuario tiene los privilegios de acceso apropiados en el computador. Una vez hecha la conexión y comprobados los privilegios del usuario, éste puede realizar la copia. La ubicación del archivo debe ser conocida para el usuario. En estos ambientes, cada computador puede estar ejecutando un SO diferente, pero como mínimo deben coincidir en el formato y significado de todos los mensajes que se puedan intercambiar.

Sistemas operativos realmente distribuidos

Los usuarios pueden acceder a los recursos remotos de la misma manera que lo hacen a los recursos locales. La migración de datos y procesos de una máquina a otra queda bajo el control del SO distribuido. Debe existir un mecanismo de comunicación global entre los procesos que permita que se puedan comunicar unos con otros. Estos mecanismos deben ser los mismos en todas las máquinas y no deben distinguirse entre comunicaciones locales y globales. Una consecuencia de ello es que se ejecutan núcleos idénticos del SO en todas las máquinas.

Las funciones, transparentes al usuario, son:

- *Migración de datos* . Dos estrategias una es transferir todo el archivo de una instalación a otra y a partir de ese momento todo el acceso a los datos es local. Otra opción es transferir sólo aquellas partes del archivo que realmente se necesitan en ese momento; si posteriormente se requieren más datos entonces se transferirá otra parte del archivo, de una forma similar a la paginación por demanda. El uso de un método u otro depende del tamaño del archivo y de la porción que se necesita.
- *Migración de cálculos* . Hay varias formas de realizar estos cálculos, mediante una *llamada a un procedimiento remoto*. Un proceso de una instalación invoca a otro procedimiento definido en otra instalación, el cual, después de ejecutarse, devuelve los resultados al proceso que lo invocó. Otra opción es mediante *mensajes*. El primer proceso envía un mensaje a la instalación donde está definido el proceso que quiere que se ejecute. Entonces, el SO de esta instalación arranca el proceso y cuando éste termina envía un mensaje con los resultados al primer proceso. En este esquema los dos procesos pueden ejecutarse concurrentemente.
- *Migración de procesos*. Un proceso puede comenzarse en una instalación, pero continuar en otra.

Las razones para la migración es que sirven para equilibrar las cargas de los sistemas, aceleran los cálculos, dividiendo un proceso en varios subprocesos que se pueden ejecutar en varias máquinas concurrentemente, las características de hardware o software de una instalación, que haga preferible que, en un determinado momento, cierto proceso continúe su ejecución en esta instalación. Hay dos formas de realizarla de forma transparente al usuario que tiene la ventaja que el usuario no necesita codificar los programas explícitamente para hacer la migración y, generalmente, se utiliza para equilibrar cargas y acelerar los cálculos entre sistemas homogéneos, o bien requiriendo que éste especifique como y a donde debe migrar el proceso. Se emplea cuando el proceso se debe trasladar por consideraciones hardware o software.

Algunas consideraciones para el diseño

Transparencia

Idealmente, un SO distribuido debería percibirse por el usuario como un sistema único, sin hacer distinción entre los recursos locales y remotos. Un sistema que cumple este objetivo se dice que es un *sistema transparente*. Una forma de conseguir la transparencia es ocultar a los usuarios la distribución. Ocultar la distribución a los programas es más difícil. Se puede hablar de:

Transparencia de la localización. Los usuarios no pueden indicar la situación de los recursos.

Transparencia de migración. Los recursos se mueven sin tener que cambiar sus nombres.

Transparencia de copia. El SO puede hacer copias de los archivos sin que lo noten los usuarios.

Transparencia con respecto a la concurrencia. Los usuarios no advierten la existencia de otros usuarios.

Transparencia con respecto al paralelismo. El sistema aprovecha las posibilidades de cálculo en paralelo sin que el programador tenga que advertirlo.

Flexibilidad

Hay dos concepciones diferentes por una parte está el modelo de sistemas con *núcleo monolítico* donde el núcleo corresponde el SO básico centralizado actual, al que se le han añadido las capacidades de red y la integración de los servicios remotos. Suelen ser extensiones de algún sistema existente, generalmente Unix. Su inconveniente es que no son muy flexibles: para instalar y depurar nuevos servicios se debe parar el sistema y arrancar otra vez el núcleo. Tienen la ventaja de tener mayor rendimiento, puesto que la realización del trabajo en el núcleo es más rápido que el envío de mensajes a los servidores remotos.

En los sistemas con micronúcleo, éste tiene los servicios mínimos, consistentes básicamente en un mecanismo de comunicación entre procesos, una gestión de la memoria, una parte de la planificación y gestión de los procesos de bajo nivel y la gestión de la entrada/salida a bajo nivel. El resto de los servicios (sistema de archivos, gestión de los procesos, manejo de las llamadas al sistema, etc.) se implementan como servidores a nivel de usuario. Es el método utilizado en la mayoría de los diseños que parten de cero. Tiene el inconveniente de un menor rendimiento, por el envío de mensajes entre los módulos.

Presenta la ventaja de ser más flexible al ser muy modular, permitiendo instalar y depurar nuevos servicios de forma fácil, puesto que no es necesario parar el sistema y arrancar otra vez el núcleo.

Fiabilidad

Tiene que ver con que si una máquina falla, otra se encargará de su trabajo, es decir, el sistema debe ser tolerante a fallos. Un sistema tolerante debe continuar su funcionamiento, aunque sea degradado, a pesar de que se produzcan fallos. La degradación puede consistir en una disminución en el rendimiento o en la funcionalidad, pero debe ser proporcional a los fallos que se produzcan.

Escalabilidad

Hace referencia a la capacidad del sistema para adaptarse a un incremento en el servicio. Un sistema escalable debe reaccionar con mayor suavidad ante un aumento de la carga que un sistema que no lo es. El rendimiento se debe degradar con mayor moderación y los recursos se deben saturar más tarde que en un sistema no escalable. El problema se resuelve añadiendo nuevos recursos, pero esto puede ocasionar una carga indirecta en otros recursos. Un sistema escalable debe tener la capacidad de crecer sin estos problemas y permitir una elevada carga de servicio, adaptándose al crecimiento del número de usuarios y admitiendo la integración de nuevos recursos adicionales de forma sencilla.

Servicios remotos

Son una forma de hacer una transferencia de datos cuando un usuario solicita un acceso a un archivo remoto. Una de las formas más comunes de servicio remoto es la *llamada a un procedimiento remoto (RPC, Remote Procedure Call)*: Cuando se invoca un procedimiento remoto, el entorno que lo invoca queda suspendido y se transfieren los parámetros a través de la red, al nodo donde el procedimiento se va a ejecutar. Cuando finaliza el procedimiento, se devuelven por la red los resultados al nodo que lo invocó.

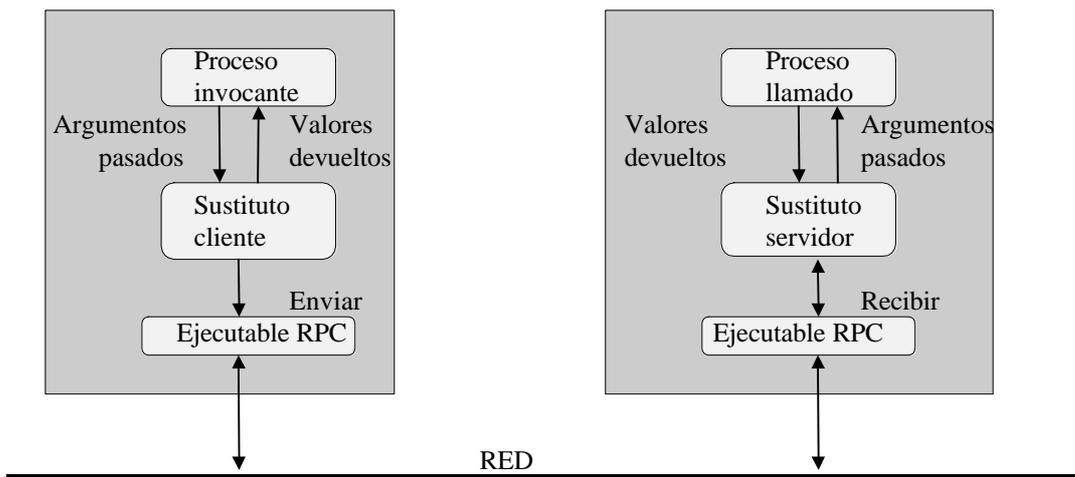
Las ventajas de las llamadas a procedimientos remotos son:

- Semántica sencilla que facilita el cálculo distribuido.
- Transparencia de la distribución para los programadores.
- Las aplicaciones para un solo computador se pueden exportar con facilidad a un entorno distribuido.
- Su generalidad, los procedimientos son uno de los mecanismos más habituales y mejor comprendidos para la comunicación entre partes de un algoritmo.

Los procedimientos remotos son apropiados para los modelos de cliente/servidor de cálculo distribuido. Las rutinas de los clientes solicitan los servicios invocando remotamente la ejecución de los procedimientos adecuados en el servidor. Por otro lado, los servidores pueden proporcionar servicios comunes por medio de los procedimientos de servidores públicos que pueden ser llamados por los clientes. En los dos entornos las rutinas públicas deben ser reentrantes o estar protegidas frente a expropiaciones mediante alguna norma de control de concurrencia.

Mecanismo de llamada a procedimientos remotos

La estructura de un programa para la llamada a procedimientos remotos se suele basar en el concepto de *sustitutos (stubs)*.



El proceso invocante hace una llamada normal, con parámetros a un procedimiento en su máquina, por ejemplo, *call p(x,y)*. Un procedimiento sustituto *p* crea un mensaje que identifica el procedimiento que se llama e incluye los parámetros. Entonces se envía este mensaje, usando una primitiva *send*. El procedimiento sustituto espera hasta que recibe la respuesta y entonces devuelve los valores recibidos al programa invocante. En la máquina remota, otro programa sustituto recibe el mensaje y realiza una llamada local *p(x,y)*. De esta manera el procedimiento se llama localmente de forma idéntica a si se tratase de un procedimiento local. El programa invocante queda suspendido hasta que el procedimiento invocado termina y

devuelve los valores. Hay una sincronía total entre el invocante y el invocado. Cuando termina la llamada el invocante continúa su ejecución. El procedimiento servidor se ejecuta en su totalidad antes de devolver el control. Las llamadas a procedimientos remotos pueden estar anidadas. Las llamadas se suelen hacer por valor, pues es muy sencillo. Por referencia, en cambio, es más difícil ya que se deben implementar punteros al sistema global.

En cuanto a la representación de parámetros y resultados en los mensajes el caso ideal es cuando el programa invocante y invocado están escritos en el mismo lenguaje, en el mismo tipo de máquina y bajo el mismo SO. En otro caso habrá diferencias en la representación de los números y de los caracteres.

Como solución si se usa un sistema completo de comunicación, esta cuestión se gestiona en la capa de presentación. Los sistemas completos tienen un coste elevado. Servicios de comunicaciones dentro de las llamadas a procedimientos remotos. Suministran un formato estándar para los objetos comunes.

Tradicionalmente las llamadas a los procedimientos remotos son síncronas que suponen que el proceso invocante debe esperar hasta recibir respuesta. La forma síncrona de implementar las RPC es fácil de entender y de programar, sin embargo, no explota completamente el paralelismo inherente de las aplicaciones distribuidas, limitando el tipo de interacciones que las aplicaciones distribuidas puedan tener. Se han desarrollado distintos servicios de RPC asíncronos manteniendo el resto de las facilidades y simplicidades de las RPC. De esta forma el proceso invocante no se bloquea, se ejecuta en paralelo al invocado.

La sincronización se puede conseguir de dos formas:

- El nivel de la aplicación en el cliente y en el servidor pueden iniciar el intercambio y verificar al final si se han realizado todas las peticiones.
- El cliente emite una cadena de llamadas asíncronas, seguida al final por una llamada síncrona. El servidor responde a la llamada síncrona sólo cuando se han completado todas las peticiones de las llamadas asíncronas precedentes.

Gestión de procesos distribuidos

Ordenación de sucesos

En muchas aplicaciones es primordial determinar el orden de ocurrencia de los sucesos. En los sistemas distribuidos esto no se puede lograr por las siguientes razones:

Retardo entre el tiempo real de ocurrencia de un suceso y el instante en el que se observa en otro sistema.

La falta de sincronización da lugar a una variación en la lectura de los relojes de los distintos sistemas.

Una forma de resolver estos problemas es mediante el método denominado de *marcación de tiempo* (time stamping, propuesto por Lamport, 1978). Esta técnica ordena los sucesos en un sistema distribuido sin usar relojes físicos. Un suceso local se puede vincular a un mensaje muy simple. Se asociarán los sucesos siempre cuando se envían los mensajes y no cuando se reciben. Se intenta ordenar los sucesos según la transmisión de mensajes. Así, en cada sistema de la red se mantiene un contador que funciona como reloj. Cada vez que se transmite un mensaje, primero se incrementa en uno el contador y a continuación se envía el mensaje con el siguiente formato:

(mensaje, marca de tiempo, identificador numérico del sistema)

Cuando se recibe un mensaje, el sistema receptor suma una al máximo entre el valor actual de su reloj y el valor de la marca de tiempo del mensaje de entrada:

$$\text{valor del reloj} = 1 + \max(\text{valor reloj actual}, \text{marca de tiempo mensaje})$$

En cada sistema, el orden de los sucesos se determina según la siguiente regla: Si A es un suceso del sistema M y B es un suceso del sistema N , se dice que A precede a B si se cumplen alguna de las siguientes condiciones:

1. La marca de tiempo de A es menor que la de B .
2. Las marcas de tiempo son iguales pero el identificador numérico de M es menor que el del sistema N .

es decir, dos sucesos de mensajes con la misma marca de tiempo se ordenan por el identificador numérico de sus sistemas.

El orden impuesto por este esquema no necesariamente debe corresponder a la secuencia de tiempo real. En este esquema lo importante no es el orden real en el que ocurren los sucesos, sino que todos los procesos coincidan con el orden impuesto a los sucesos. Esta técnica es sencilla y resulta muy eficaz, por lo que se utiliza en muchos algoritmos para la exclusión mutua y el bloqueo.

Exclusión mutua

Los algoritmos pueden ser centralizados o distribuidos, en el caso completamente centralizado sólo un nodo coordinador toma las decisiones sobre la asignación de los recursos, toda la información necesaria está concentrada en el nodo coordinador. Es sencillo y fácilmente se ve como se impone la exclusión mutua. Su inconveniente es que si falla el nodo coordinador, el mecanismo de exclusión mutua se viene abajo, además toda la asignación y desasignación de recursos requiere un intercambio de mensajes con el nodo coordinador forma el cuello de botella del sistema.

Los algoritmos distribuidos se caracterizan por:

1. En promedio, todos los nodos tienen una cantidad igual de información.

2. Cada nodo sólo tiene una representación parcial del sistema total y debe tomar las decisiones basándose sólo en esta información.
3. Todos los nodos tienen la misma responsabilidad en la decisión final.
4. En promedio, todos los nodos asignan el mismo esfuerzo en efectuar una decisión final.
5. En general, el fallo de un nodo no colapsa al sistema completo.
6. No existe un único reloj global para regular el tiempo de los sucesos. Se ordenan los sucesos tal como se vio anteriormente.

Algunos algoritmos distribuidos para implementar la exclusión mutua son las colas distribuidas y el paso de testigos.

Colas distribuidas

Requiere la existencia de un orden total de todos los sucesos del sistema (se puede utilizar el algoritmo de marcación de tiempo). Cuando un proceso P_i quiere entrar en su región crítica envía un mensaje de solicitud, *solicita* (P_i, T_i, i) donde T_i es la marca de tiempo e i es el número del sistema, a todos los procesos del sistema (incluyéndose a sí mismo). Al recibir un mensaje de solicitud, un proceso puede responder de inmediato, es decir, enviar un mensaje de respuesta a P_i , o puede diferirla, por ejemplo porque ya se encuentra en su región crítica. Un proceso que ya ha recibido respuesta de todos los demás puede entrar en su región crítica, colocando en una cola todas las solicitudes que reciba y posponiendo su contestación. Después de salir envía los mensajes de respuesta a todas las peticiones que difirió. La decisión de si un proceso P_i responde de inmediato a un mensaje de solicitud o difiere su respuesta, depende de:

- a. Si el proceso P_i se encuentra en su región crítica, difiere su respuesta.
- b. Si el proceso P_i no espera entrar en su región crítica, entonces envía de inmediato su respuesta.
- c. Si el proceso P_i espera entrar en su región crítica, pero aún no lo ha hecho, entonces compara la marca de tiempo de su solicitud con la de la solicitud efectuada por P_j . Si su propia marca de tiempo es mayor, envía inmediatamente su respuesta, si no la difiere.

En este método el número de mensajes por entrada a la región crítica es $2 \cdot (n-1)$, donde n es el número de nodos del sistema. Sus inconvenientes son:

- Cada proceso necesita conocer la identidad de todos los demás. Cuando entra un nuevo proceso, hay que distribuir su nombre entre los demás y éste a su vez debe conocer la identidad de los otros. Esta tarea no es sencilla, ya que pueden estar circulando mensajes de solicitud y de respuesta cuando se incluye este nuevo proceso.
- Si falla un proceso, el esquema se viene abajo. Se puede solucionar supervisando continuamente el estado de todos los procesos. Si uno falla, se notifica a los demás para que no le envíen mensajes.
- Los procesos que no han entrado en su región crítica deben realizar pausas frecuentes para responder a los restantes que quieren acceder a su región crítica.

Este esquema resulta adecuado para pequeños conjuntos estables de procesos cooperativos.

Paso de testigo

Consiste en hacer circular un *testigo* entre los procesos. El testigo es un tipo especial de mensaje que pasa de un proceso a otro en el sistema, de forma que su posesión permite entrar en la región crítica. Cuando el proceso sale de su región crítica pasa el testigo al siguiente proceso. Se necesitan dos estructuras de datos, una para el testigo que se pasa de proceso en proceso, y es un array (*testigo*) cuyo k -ésimo elemento es la marca de tiempo de la última vez que el testigo visitó al proceso P_k .

Otra es que cada proceso mantiene un array de peticiones (*petición*), cuyo j -ésimo elemento es la marca de tiempo de la última petición recibida de P_j .

El algoritmo actúa de la siguiente forma:

1. Inicialmente, el testigo se asigna arbitrariamente a un proceso.
2. Cuando un proceso quiere entrar en una región crítica, entra si tiene el testigo, si no transmite un mensaje de petición de marca de tiempo a los otros procesos y espera hasta tener el testigo.
3. Si el proceso P_i sale de su región crítica, debe transmitir el testigo a otro proceso que se elige buscando en el array de peticiones la primera entrada j , tal que la marca de tiempo de la última petición del testigo de P_j , (*petición*(j)), sea mayor que el valor que se tiene para P_j correspondiente a la última vez que tenía el testigo (*testigo*(j)). Es decir, se ha de verificar que *petición*(j) > *testigo*(j).
4. No se permite entrar a una segunda región crítica con el mismo testigo.

Sólo un proceso puede estar en la sección crítica (sólo hay un testigo) y, además, si el recorrido es en forma de anillo unidireccional, se asegura que no se presentan bloqueos indefinidos. El algoritmo requiere, cuando no se tiene el testigo n mensajes; y ningún mensaje si tiene el testigo. Hay dos posibles fallos que se deben considerar:

- a. Si el testigo se pierde es necesario hacer una elección para generar uno nuevo.
- b. Si falla un proceso se debe establecer un nuevo anillo lógico.

Bloqueos

Son similares a los bloqueos en los sistemas con un único procesador, aunque las situaciones creadas pueden ser más graves y los problemas más complejos. Su manipulación en un sistema distribuido es más compleja porque ningún nodo tiene un conocimiento preciso del estado actual del sistema completo, y los mensajes entre los procesos involucran un retardo que es impredecible. Se usan dos tipos de bloqueos el *bloqueo de recursos* que ocurre cuando los procesos compiten por el acceso exclusivo a un recurso que está retenido por otro proceso; y el *bloqueo de comunicación* que ocurre cuando cada uno de los procesos espera la llegada de un mensaje y ninguno de ellos lo envía.

Al igual que en los sistemas monoprocesadores, para evitar los interbloqueos en los sistemas distribuidos se pueden utilizar las técnicas de prevención y de evitación, o bien permitir que ocurran y entonces detectarlos y recuperarse de ellos, o simplemente ignorarlos (esta última es muy popular). No se suele tener un sistema de interbloqueo global, sino disponer de mecanismos individuales para cada sistema o aplicación. También se emplea la detección y recuperación de los bloqueos, debido principalmente a que es muy difícil prevenirlos y evitarlos.

Prevención

Se pueden utilizar los algoritmos de prevención para sistemas monoprocesadores con las modificaciones adecuadas, la condición de *espera circular* puede ser prevenida definiendo un orden lineal de los tipos de recursos. Si un proceso tiene asignado un determinado tipo de recurso, entonces sólo puede pedir recursos de los tipos siguientes en el orden establecido. El inconveniente del método es que los recursos no se piden en el orden en el que se usan, pudiéndose retener recursos durante mucho tiempo innecesariamente.

La condición de *retención y espera* se puede prevenir haciendo que la petición de los recursos que necesita un proceso se haga de una sola vez, bloqueándolo hasta que se puedan garantizar simultáneamente todas las peticiones de recursos. El inconveniente del método es que el proceso puede tener que esperar mucho tiempo antes de tener todos los recursos, cuando posiblemente podría seguir su ejecución con sólo algunos. Además, algunos recursos asignados a un proceso puede que pasen mucho tiempo sin ser usados.

Los métodos anteriores requieren que los procesos determinen previamente sus necesidades de recursos, lo cual no siempre es posible. Se han ideado algoritmos para prevenir los interbloqueos en sistemas distribuidos sin este requerimiento. Dos de estos algoritmos basados en el uso de marcas de tiempo se deben a Rosenkrantz (1978), el Algoritmo de espera-muerte (*wait-die*) y el Algoritmo de herida-espera (*wound-wait*); para ello se precisa una bastracción a mayor nivel la *transacción o transacción atómica*. En este esquema se tiene en cuenta la transacción entre procesos que tienen que ponerse de acuerdo en la operación a realizar aceptándola. El proceso iniciador anuncia que quiere que el resto de los procesos se comprometan con el trabajo realizado hasta entonces. Si todos aceptan, los resultados son permanentes. Pero si uno o más procesos fallan antes de expresar se acuerdo, entonces la situación vuelve al estado anterior al del comienzo de la transacción, sin que existan efectos colaterales. Cada transacción lleva la marca de tiempo de su creación durante toda su vida.

Algoritmo de espera-muerte

Si $T1$ tiene un recurso R que solicita $T2$, la asignación del recurso se hace de la siguiente forma:

```
if  $m(T2) < m(T1)$  then
    se interrumpe  $T2$  (espera)
else
    aborta  $T2$  (muerte)
endif
```

Es decir, la transacción que solicita el recurso se bloqueará si su marca de tiempo es más antigua que la del que tiene el recurso; o será abortada si su marca de tiempo es más moderna. En este último caso a pesar de que la transacción $T2$ vuelva a arrancar conservará su marca de tiempo original, con lo que podrá “envejecer” y así a la larga acceder al recurso. En cualquier caso, no hay expropiación: $T1$ conserva el recurso hasta que termina de utilizarlo.

Algoritmo de herida-espera

Si $T1$ tiene un recurso R que solicita $T2$, la asignación del recurso se hace de la siguiente forma:

```
if  $m(T2) < m(T1)$  then
    aborta  $T1$  (herido)
else
    se interrumpe  $T2$  (espera)
endif
```

En este algoritmo sí existe expropiación, si la marca de tiempo de la transacción solicitante es más antigua que la del que lo retiene, se aborta esta última (al igual que antes la transacción abortada conserva su marca de tiempo original). En otro caso la transacción solicitante se pasa a estado de espera.

Detección

A los procesos se les permite que obtengan libremente los recursos que necesitan. Es después de que ocurre un interbloqueo cuando se determina su existencia. Cuando se detecta un interbloqueo, se selecciona uno de los procesos involucrados para que libere el recurso necesario para romper el bloqueo. Hay diferentes esquemas para resolver el problema:

El *enfoque centralizado* donde un nodo es el responsable de la detección de los bloqueos. Todos los mensajes de solicitud y liberación de recursos se envían a un proceso coordinador, además de transmitirlo al proceso que controla ese recurso, tiene el inconveniente de que si el nodo coordinador falla, el sistema se viene abajo. Este nodo puede convertirse en

un cuello de botella del sistema a causa de la gran cantidad de mensajes que debe gestionar. La ventaja es que los algoritmos son sencillos y al estar toda la información en un único sistema los bloqueos se pueden resolver de forma óptima.

En un *enfoque jerárquico* los nodos se organizan en una estructura de árbol. Cada nodo tiene información de los recursos asignados a los nodos que dependen de él, por lo que se pueden detectar los bloqueos en los niveles inferiores. Tiene la dificultad de configurar el sistema para poder localizar los posibles bloqueos. Como ventaja es que no es tan vulnerable a los fallos de un nodo.

En un *enfoque distribuido* todos los nodos cooperan para detectar los bloqueos. Esto supone un gran intercambio de información con las marcas de tiempo correspondientes. La desventaja es que los algoritmos para la detección son laboriosos y difíciles de diseñar, ya que varios nodos pueden detectar el mismo bloqueo sin darse cuenta que hay también otros nodos implicados. Sin embargo no es vulnerable a los fallos que se produzcan de un nodo y ningún nodo está sobrecargado con los algoritmos de detección.

Como método de detención de bloqueos se puede usar el algoritmo de Chandy. En él se permite a los procesos que soliciten varios recursos al mismo tiempo, de esta forma un proceso puede esperar dos o más recursos simultáneamente. Actúa de la siguiente forma:

Un proceso que espera un recurso envía un mensaje especial de exploración al proceso que tiene los recursos que necesita. Este mensaje consta de tres partes: el indicativo del proceso recién bloqueado, el indicativo del proceso que envía el mensaje y; el indicativo del proceso receptor del mensaje (P1, P1, P2). El proceso que recibe el mensaje lo reenvía sólo cambiando la segunda y tercera parte del mensaje (el indicativo del emisor y el del receptor), al proceso o procesos de los que espera un recurso si los hay. No varía la primera parte del mensaje que contiene el indicativo del proceso emisor (P1,P2,Px). Este proceso se repite sucesivamente. Si un proceso bloqueado recibe un mensaje cuya primera parte es su propio indicativo, significa que se ha completado un ciclo y por lo tanto que existe un bloqueo.

El problema que surge una vez comprobado que existe un bloqueo es decidir que proceso se aborta: el que ha iniciado el ciclo o el de indicativo mayor. Es un problema sin una solución óptima conocida ya que las soluciones teóricas que se presentan como óptimas, no son tales en la realidad práctica.

Sistemas de archivos compartidos

Conceptos previos:

Servicio: entidad software que se puede ejecutar en una o más máquinas a petición de un cliente, no conocido a priori, al que se le ofrece algún tipo particular de funcionalidad.

Servidor: software que ofrece un servicio y que se ejecuta en una máquina.

Cliente: proceso que solicita un servicio mediante una serie de operaciones a través del interfaz con el cliente.

Un sistema de archivos ofrece un servicio de archivos a los clientes.

Formas de implementar un sistema de archivos distribuido en algunas configuraciones hay máquinas dedicadas a operar sólo como servidores de archivos, en otras las máquinas pueden ser a la vez servidores y clientes. Por otro lado el sistema de archivos distribuidos puede estar implementado como parte del SO, o puede ser una capa de software que gestiona la comunicación entre los sistemas operativos y los sistemas de archivos convencionales.

Su objetivo básico es parecer a sus clientes un sistema de archivos convencional: el usuario no debe distinguir entre los archivos locales y los remotos. El principal problema es conseguir que el tiempo necesario para satisfacer las solicitudes de servicio sean equiparables a los de un sistema convencional. Este tiempo, además de los tiempos de un sistema convencional (tiempo de acceso al disco y tiempo de procesamiento), está compuesto por el tiempo de petición de la solicitud a un servidor, el tiempo de respuesta al cliente y el tiempo de procesamiento de los protocolos de comunicaciones.

Un sistema comercial de archivos distribuidos: NFS

Es el *Sistema de archivos en red* (NFS, *Network File System*) de Sun Microsystems. Generalmente todos los computadores que comparten el sistema de archivos mediante NFS están en la misma red local, sin embargo, es posible implementarlo en redes de área extensa. Se consideran computadores clientes y servidores (soportan el sistema de archivos). Un mismo computador puede ser servidor y cliente a la vez.

Cada uno de los servidores NFS *exporta* uno o varios de sus directorios para que los clientes remotos puedan acceder a los mismos. Los clientes pueden montar el directorio exportado por el servidor como un directorio más en su jerarquía local de directorios. A partir de ese momento se comporta para el cliente como un directorio local más. Si dos o más clientes montan el mismo directorio al mismo tiempo, pueden compartir archivos en sus directorios comunes.

Protocolos NFS

Se hace uso de dos protocolos el primer protocolo de NFS gestiona el montaje. Un cliente puede enviar el nombre de una ruta de acceso a un servidor y solicitar el permiso para montar ese directorio en su jerarquía de directorios. Si el nombre de la ruta de acceso es válido y ese directorio ha sido previamente exportado por el servidor, el servidor devuelve un

manejador de archivo (file handle) al cliente con las informaciones relativas al directorio. Las llamadas posteriores usarán ese manejador.

Hay dos formas de realizar el montaje, en el *montaje automático* el montaje de ciertos directorios se realiza automáticamente cuando se arranca el cliente. Una alternativa es el *automontaje* (Sun). Se permite que una serie de directorios remotos estén asociados con un directorio local. La primera vez que se abre un archivo remoto, automáticamente se monta el directorio correspondiente. Se evita el montaje de directorios que a lo mejor no se van a utilizar en esa sesión.

El segundo protocolo es el de acceso a los directorios y archivos. NFS soporta la mayoría de las llamadas al sistema Unix, con las excepción de *open* y *close*, de forma que no es necesario abrir un archivo antes de leerlo, por ejemplo. Esto además hace que, si el servidor falla, cuando vuelve a arrancar no se pierde la información acerca de los archivos abiertos, puesto que estos no existen.

Arquitectura NFS

Consiste en tres capas:

- a. El interfaz del sistema de archivos Unix, en las que se soportan las llamadas al sistema (*open*, *read*, *write*, *close*).
- b. El *sistema virtual de archivos* (VFS, *Virtual File System*). Maneja una tabla con una entrada por cada archivo abierto. VFS distingue entre los archivos locales y los remotos.
- c. La capa de servicios NFS. Implementa el protocolo NFS.

La forma de manipulación de un archivo remoto es:

1. El cliente inicia la operación con una llamada normal al SO; la capa de llamadas al sistema la transforma en una operación VFS para el *nodo-v* adecuado.
2. En la capa VFS se identifica al archivo como un archivo remoto y se invoca el procedimiento VFS correspondiente.
3. Se envía una llamada RPC a la capa de servicio NFS del servidor remoto, donde se introduce en la capa VFS; en este caso se observa que es una llamada local y se invoca la operación correspondiente del sistema de archivos.
4. Los resultados se devuelven siguiendo el camino inverso.

El sistema de archivos Andrew

El *sistema de archivos Andrew* (AFS, *Andrew File System*) tiene como principal característica su capacidad de crecimiento, puesto que el objetivo del sistema es abarcar más de cinco mil estaciones de trabajo. AFS establece una diferencia entre las máquinas cliente (estaciones de trabajo) y las máquinas servidoras dedicadas. Tanto los clientes como los servidores ejecutan Unix y están conectados por una red compuesta a la vez por redes locales. Los servidores dedicados, que se denominan colectivamente *Vice*, presentan a los clientes el espacio de nombres compartidos como una jerarquía de archivos homogénea, idéntica y transparente a la ubicación. Las estaciones de trabajo ejecutan el protocolo *Venus* para comunicarse con *Vice*.

Cada agrupamiento (red local) consiste en un conjunto de estaciones de trabajo y un representante de *Vice*, denominado *servidor del agrupamiento*. Las estaciones de trabajo deben utilizar lo más posible el servidor de su propio agrupamiento, de forma que las referencias entre agrupamientos sean lo menos frecuentes posibles. Los diferentes agrupamientos se hallan conectados a la red general utilizando encaminadores. Este sistema de agrupamiento permite una fácil ampliación de la red. Además, para descargar de trabajo a los servidores, los archivos se colocan completos en memoria caché (en el disco local) como mecanismo clave para las operaciones con archivos remotos.

8. EJEMPLOS DE SISTEMAS OPERATIVOS

UNIX

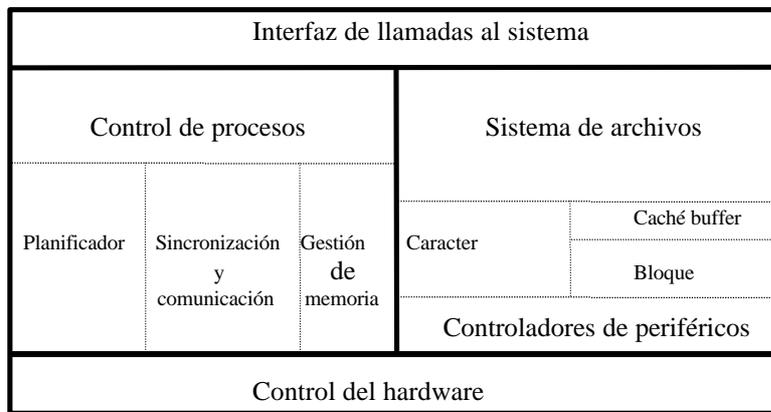
Descripción

UNIX es un sistema operativo multiusuario de tiempo compartido interactivo, escrito en lenguaje C. El hardware está rodeado por el núcleo de Unix, que es el auténtico Sistema Operativo, se denomina así (núcleo) para enfatizar su aislamiento de las aplicaciones y de los usuarios. Le sigue la capa de librerías, que contiene un procedimiento para cada llamada al sistema. Por encima de estas capas, todas las versiones de Unix proporcionan una serie de programas de utilidades.

Se pueden considerar tres interfaces distintas:

- La interfaz de llamadas al sistema.
- La interfaz de librería.
- La interfaz del usuario, que está formada por los programas de utilidades estándar.

El auténtico Sistema Operativo es el núcleo, cuya estructura es:



HARDWARE

Interactuando con el hardware hay unas rutinas primitivas, que corresponden al bloque de control del hardware. El interfaz de llamadas al sistema permite al software de alto nivel tener acceso a funciones específicas del núcleo. El resto del núcleo se puede dividir en dos partes, la de control de procesos y la de Entrada/Salida.

El control de procesos se responsabiliza de la planificación y distribución de los procesos y de la sincronización y comunicación de éstos con la gestión de memoria. La gestión de archivos intercambia datos entre memoria y los dispositivos externos, tanto de cadena de caracteres como de bloques.

Control y sincronización de procesos

En Unix todos los procesos, excepto dos procesos básicos del sistema, se crean mediante órdenes del programa de usuario. Los dos procesos especiales son:

- El proceso 0, que se crea cuando el sistema se arranca y que corresponde al proceso intercambiador.
- El proceso 1, que es creado por el proceso 0. Es el proceso *init*, que es el antecesor de todos los procesos. Cuando se conecta un usuario al sistema, el proceso 1 crea un proceso de usuario para él. El programa del usuario puede crear procesos hijos, de forma que una aplicación pueda consistir en varios procesos relacionados.

Los posibles estados de los procesos en Unix son:

- Ejecución en modo usuario.
- Ejecución en modo núcleo.
- Preparado para ejecución en memoria principal, tan pronto como lo planifique el núcleo.
- Espera en memoria principal. El proceso está en memoria principal esperando a que ocurra un suceso.
- Preparado para ejecución pero en memoria secundaria. El intercambiador (proceso 0) debe transferirlo a la memoria principal antes de que el núcleo pueda planificar su ejecución.
- Espera en memoria secundaria. El proceso está esperando a que ocurra un suceso y tiene que intercambiarse desde la memoria secundaria.

7. Adelantado. El proceso se está ejecutando desde el modo núcleo al modo usuario, pero el núcleo lo adelanta y hace un cambio de contexto para planificar otro proceso. Esencialmente es lo mismo que el estado 3, la distinción se hace para enfatizar la forma de entrar en el estado adelantado.
8. Creado o nonato. El proceso se ha creado de nuevo y está en un estado de transición; el proceso existe pero aún no está preparado para ejecución. Este es el estado inicial para todos los procesos, excepto el proceso 0.
9. Zombi. El proceso ejecuta la llamada *exit* al sistema y queda en un estado zombi. El proceso no existe, pero libera un registro para unir a su proceso padre conteniendo un código de salida y algunas estadísticas. Es el estado final de un proceso.

La imagen de un proceso se organiza como la unión de tres partes:

- a. *Contexto a nivel de usuario*. Contiene los elementos básicos de un programa de usuario. El programa se divide en áreas de texto que contiene las instrucciones del programa y es de sólo lectura y áreas de datos. Además se dispone de una pila de usuario y de la memoria compartida que ocupa el espacio de direcciones virtuales del proceso.
- b. *Contexto de registro*. Almacena la información del estado de un proceso cuando no se está ejecutando. Consta de las siguientes componentes:
 - El contador del programa
 - El registro de estado del proceso
 - El puntero de la pila
 - Registros de propósito general, que contienen datos generados por el proceso durante su ejecución.
- c. *Contexto a nivel de sistema*. Contiene el resto de información que el SO necesita para gestionar el proceso. Consta de una parte estática, cuyo tamaño es fijo y permanece con el proceso a lo largo de la vida de éste, y una parte dinámica, que varía en tamaño a lo largo de la vida del proceso. El contexto a nivel de sistema tiene los siguientes componentes:
 - La *entrada a la tabla del proceso*. Define el estado del mismo y contiene información de control que siempre está accesible al SO.
 - El *área U* (usuario). Contiene información de control del proceso que necesita ser accedida sólo en el contexto del proceso.
 - *Tablas de páginas y regiones*. Definen la correspondencia entre las direcciones virtuales y las físicas, además de campos de permisos que indican el tipo de acceso permitido. Se utilizan por el gestor de memoria.
 - *Pila del núcleo*. Contiene los marcos de la pila de los procedimientos del núcleo como un proceso ejecutado en modo núcleo.
 - La *parte dinámica del contexto a nivel de sistema*. Consiste en un conjunto de capas, visualizadas como una pila. Cada una contiene la información necesaria para recuperar la capa anterior, incluyendo el contexto de registro.

Creación de procesos

Se realiza mediante una llamada del sistema *fork*. La sintaxis para *fork* es:

```
pid=fork();
```

Cuando un proceso hace una petición *fork*, el núcleo realiza las siguientes funciones:

1. En la tabla de procesos se le asigna un hueco al nuevo proceso.
2. Al proceso hijo se le asigna un único identificador (*ID*) de proceso.
3. Se hace una copia de la imagen del proceso padre, con excepción de la memoria compartida.
4. Se incrementan los contadores de los archivos propiedad del padre, ya que pertenecen a otro proceso más.
5. Al proceso hijo se le asigna un estado de “preparado para ejecución”.
6. Se devuelve al proceso padre el número *ID* del hijo, y un valor 0 al hijo.

Todo este proceso se realiza en modo núcleo en el proceso padre. Cuando el núcleo termina estas funciones, el distribuidor puede:

- a. Permanecer en el proceso padre. El control vuelva al padre en modo usuario.
- b. Transferir el control al proceso hijo.
- c. Transferir el control a otro proceso.

Comunicación entre procesos

En Unix se distinguen distintos algoritmos para la comunicación y sincronización entre procesos:

- a. *Tubos* (pipes). Permiten transferir datos entre los procesos y sincronizar la ejecución de los mismos. Usan un método de productor-consumidor (en lectura/escritura), y una cola donde cada proceso escribe y el otro lee. Los tubos se crean con un tamaño fijo. Hay dos clases de tubos:

Tubos no etiquetados. Sólo se pueden comunicar procesos relacionados (un proceso con sus descendientes).

Tubos etiquetados. Se pueden comunicar procesos no relacionados.

- b. *Señales.* Son un mecanismo software que informan a un proceso de que ha ocurrido algún suceso asíncrono. Son similares a las interrupciones. Los procesos pueden enviar señales a los otros con la llamada al sistema *kill*, o bien el núcleo las puede enviar internamente.
- c. *Mensajes.* Permiten a los procesos enviar cadenas de datos formateados. En Unix a cuatro llamadas al sistema para los mensajes:
 - msgget.* Devuelve un descriptor del mensaje que designa a una cola del mensaje para usar en otras llamadas al sistema.
 - msgctl.* Tiene las opciones de poner y devolver parámetros asociados con un descriptor del mensaje y una opción para eliminar descriptores.
 - msgsnd.* Envía un mensaje.
 - msgrcv.* Recibe un mensaje.
- d. *Memoria compartida.* Es, quizás, la forma más rápida de comunicación entre procesos, posibilita que los procesos compartan parte de su espacio de direcciones virtuales. Los procesos leen y escriben en memoria compartida con las mismas instrucciones máquina que utilizan en otras partes de su espacio de memoria. Se usan llamadas al sistema similares a las usadas para los mensajes:
 - shmget.* Crea una nueva región de memoria compartida o devuelve una existente.
 - shmat.* Une una región con el espacio de direcciones virtuales de un proceso.
 - shmdt.* Adhiere una región al espacio de direcciones virtuales de un proceso.
 - shmdt.* Elimina esta región del espacio de direcciones virtuales del proceso.
 - shmctl.* Manipula varios parámetros asociados a la memoria compartida.
- e. *Semáforos.* Se crean en conjuntos (un conjunto consta de uno o más semáforos). Constan de los siguientes elementos:
 - El valor del semáforo.
 - El *ID* (identificador) del último proceso que manipuló al semáforo.
 - El número de procesos que esperan que el valor del semáforo aumente.
 - El número de procesos que esperan que el valor del semáforo sea igual a 0.

Las llamadas al sistema de semáforos son:

Semget: Crea y da acceso a un conjunto.

Semutl: Operaciones de control del conjunto.

Semop: Manipulación de valores de los semáforos. Su sintaxis es:

oldval = semop (id, oplist, count)

id es el descriptor devuelto por *semget*; *oplist* es un puntero a un array de operaciones del semáforo y *count* es el tamaño del array. *oldval* es la última operación del semáforo.

Gestión de memoria

En las primeras versiones UNIX utilizaba particiones variables de memoria, en la actualidad hace uso de memoria paginada. En Unix System V se emplean unas estructuras de datos que son independientes de la máquina estas estructuras son:

Se tiene una *tabla de páginas* para cada proceso. Con una entrada para cada una de las páginas de memoria virtual del proceso. Cada entrada consta de siete campos:

1. El número del marco en la memoria virtual.
2. La edad. Tiempo que ha estado la página en la memoria sin referenciarse.
3. Copias para escritura. Si un proceso escribe en una página compartida, se hace primero una copia de la página para los otros procesos.
4. Bit de página modificada. Indica si la página ha sido modificada.
5. Bit de referencia. Indica si la página ha sido referenciada.
6. Bit de validación. Indica si la página está en memoria principal.
7. Bit de protección. Indica si se permiten operaciones de escritura.
8. El descriptor de bloques del disco. Describe la copia en el disco de la página virtual. Posee una entrada por cada página en la tabla de páginas. Se pueden resaltar tres campos del descriptor:
 - a. El número del dispositivo de intercambio que corresponde al número lógico del dispositivo que tiene la página correspondiente. Así se pueden usar más de un dispositivo para intercambio.
 - b. El número de bloque donde está la página en el dispositivo de intercambio.
 - c. El tipo de almacenamiento: unidad de intercambio o archivo ejecutable.

La *tabla de datos del marco de página* describe cada uno de los marcos de la memoria real. Los marcos disponibles se enlazan juntos en una *lista de marcos libres*. Entre las entradas a esta tabla se encuentran:

- El estado de la página. Indica si este marco está libre o tiene una página asociada.
- Un contador de referencia con el número de procesos que referencia la página.
- El dispositivo lógico que contiene una copia de la página.
- El número del bloque en el que está la copia de la página en el dispositivo lógico.
- Un puntero a otra entrada de esta tabla, a una lista de páginas libres y a una cola *hash* de páginas.

La *tabla del intercambiador* tiene una entrada por cada página en el dispositivo y existe una por cada dispositivo de intercambio. Esta tabla tiene dos entradas:

1. El contador de referencia. Número de puntos de entradas de la tabla de páginas a una página en el dispositivo de intercambio.
2. El identificador de la página en la unidad de almacenamiento.

Sistemas de archivos

En UNIX se distinguen cuatro tipos de archivos:

Ordinarios. Contienen la información del usuario, los programas o las utilidades del sistema.

Directorios. Contienen listas de nombres de archivos, más los punteros asociados a los *nodos-i*.

Especiales. Corresponden a los periféricos: impresoras, discos, etc.

Etiquetados. Corresponden a los *tubos etiquetados*.

Todos los archivos se gestionan por el SO mediante *nodos-i*. Los archivos se ubican dinámicamente, sin usar una preubicación. Así los bloques de un archivo en el disco no son necesariamente contiguos. Para mantener pequeña la estructura de *nodos-i* se utiliza la indexación en varios niveles para los archivos grandes y referencia directa para los archivos pequeños. La ventaja de este sistema es que los *nodos-i* son relativamente pequeños y de tamaño fijo, de forma que se pueden mantener en memoria principal, y que los archivos que ocupan poco espacio se pueden acceder sin indexación.

Subsistema de entrada/salida

En UNIX todos los periféricos están asociados a un archivo especial que se gestiona por el sistema de archivos. Se consideran dos tipos de periféricos: de *bloque* y de *carácter*. Los periféricos de bloque, son periféricos de almacenamiento de acceso arbitrario. Los orientados a caracteres incluyen otros tipos (impresoras, terminales). La E/S se puede realizar utilizando un buffer. Hay dos mecanismos de buffer:

1. *Sistema de caché*. Es esencialmente un caché de disco. Las transferencias entre la caché del buffer y el espacio de E/S se realiza mediante DMA, ya que ambos están en memoria principal.
2. *Colas de caracteres*. Resulta apropiado para los periféricos orientados a caracteres. Se utiliza el modelo productor-consumidor.

La E/S sin buffer es simplemente una operación de DMA entre el periférico y el espacio de memoria del proceso. Este es el método más rápido de realizar una entrada/salida, sin embargo, un proceso que esté realizando una transferencia de entrada/salida sin buffer está bloqueado en la memoria principal y no puede intercambiarse.

WINDOWS NT

Nace a causa de la necesidad de explotar las capacidades de los nuevos microprocesadores de 32 bits. Es un sistema monousuario multitarea. Este enfoque se debe a las siguientes razones:

1. Al aumento de velocidad y capacidad de memoria de los microprocesadores.
2. Al soporte de memoria virtual.
3. Al aumento de las aplicaciones cliente/servidor en redes locales.

Descripción

Esta inspirada en el sistema operativo Mach, que se diseñó con el objetivo de competir directamente con Unix, incluyendo en su núcleo gran parte de Unix 4.3 BSD. Windows NT tiene una estructura altamente modular, lo que le da una gran flexibilidad. Además, se puede ejecutar en una gran cantidad de plataformas y soporta aplicaciones escritas para otros sistemas operativos. Al igual que en otros sistemas operativos, en Windows NT se distingue entre el software orientado a aplicaciones (que se ejecuta en *modo usuario*) y el software del SO (que se ejecuta en *modo núcleo* o *modo privilegiado*). Este último tiene acceso a los datos del sistema y al hardware, mientras que el resto del sistema, ejecutado en modo usuario, lo tiene limitado.

El sistema operativo se divide en cuatro capas:

1. *Capa de abstracción del hardware* (HAL, *Hardware Abstraction Layer*). El objetivo de esta capa es que el núcleo vea por igual a todo el hardware, independientemente de la plataforma que se use. Para ello

realiza la conversión entre las órdenes y las respuestas del hardware genérico y el de una plataforma específica.

2. *Núcleo* (kernel). Es la parte central del SO y se encarga de gestionar la planificación y conmutación de contexto, los manipuladores de excepciones y de interrupciones y la sincronización.
3. *Subsistemas*. Se incluyen los módulos para funciones específicas que hacen uso de los servicios básicos dados por el núcleo. Estos módulos son: el *gestor de objetos*, el *gestor de memoria virtual*, el *gestor de procesos*, el *monitor de referencia de seguridad* y las *facilidades de llamada a los procedimientos locales*. Hay un subsistema, el *gestor de E/S* (en el que se incluye el sistema de archivos, el gestor de caché, los manejadores de dispositivos y de red, etc.), que evita que el núcleo, interactúe directamente con el hardware, esto se requiere por razones de eficiencia y, también en parte, por las operaciones de E/S.
4. *Servicios del sistema*. Esta es la última capa y proporciona un interfaz para el software que se ejecuta en modo usuario.

Por encima de estas capas están los *subsistemas protegidos*, que se encargan de la comunicación con el usuario final. Para soportar esta estructura, Windows NT utiliza un modelo cliente/servidor pero aplicado a un único sistema. Cada servidor se implementa como uno o más procesos, de forma que cada proceso espera que algún cliente (un programa de aplicación u otro módulo del SO) realice una petición de alguno de sus servicios. Esta petición se hace enviando un mensaje al servidor que realiza la operación pedida y contesta mediante otro mensaje.

Este tipo de arquitectura cliente/servidor simplifica mucho el SO básico puesto que no proporciona ningún API (*Application Programming Interface*), se pueden construir otros API sin ningún conflicto, además de ser una forma natural para el cálculo distribuido. Además, también se aumenta la fiabilidad, puesto que cada servidor se ejecuta en un proceso separado con su propia partición de memoria y protegido de otros servidores, de forma que si falla no corrompe al resto del sistema.

Otro aspecto importante de Windows NT es su soporte de *hebras* dentro de los procesos. En Windows NT un proceso es un conjunto de una o más hebras junto con los recursos del sistema asociados. Esto corresponde al concepto de programa en ejecución. Otra de las características de Windows NT es la utilización de los conceptos de diseño orientado a objetos. Aunque no es un SO orientado a objetos puro, ya que no está implementado en un lenguaje orientado a objetos y no incorpora la herencia y el polimorfismo, pero sí incorpora la encapsulación y las clases e instancias de objetos.

Control y sincronización de procesos

El diseño de los procesos de Windows NT está marcado por la necesidad de proporcionar soporte para diferentes entornos de sistemas operativos. Por eso los servicios proporcionados por el núcleo del NT son relativamente sencillos y de propósito general. Esto permite que cada subsistema del SO emule una funcionalidad y estructura de procesos particular. Entre las características de los procesos de NT se pueden resaltar:

1. Los procesos NT son implementados como objetos.
2. Un proceso ejecutable puede tener una o más hebras.
3. Tanto los objetos de los procesos como los de las hebras llevan incorporadas las capacidades de sincronización.
4. El núcleo de NT no mantiene ninguna relación con los procesos que crea, incluidas las relaciones padre – hijos.

La concurrencia entre los procesos se consigue porque hebras de diferentes procesos se pueden ejecutar concurrentemente. Además si se dispone de varias UCP's se pueden asignar distintos procesadores a hebras del mismo proceso, de forma que se ejecuten concurrentemente. Para implementar los mecanismos de sincronización entre las hebras se tiene la familia de objetos de sincronización que son, entre otros: procesos, hebras, archivos, sucesos, semáforos y relojes. Los tres primeros objetos tienen otros usos, pero también se pueden utilizar para sincronización, el resto de los tipos de objetos se diseñan específicamente para soportar la sincronización.

Gestión de memoria

Windows NT se diseñó para implementarlo en diferentes tipos de procesadores, y en uno en los que más se pensó fue el Intel 80486, por ello se adoptó el tamaño de página de 4 Kbytes del 80486 como base de su esquema de memoria virtual. En el 80386 y en el 80486 se incluye un hardware especial para soportar la segmentación y la paginación. Aunque estas funciones es posible desactivarlas. Cuando se emplea segmentación, cada dirección virtual (o lógica) se compone de un campo de referencia al segmento de 16 bits (2 de ellos destinados a los mecanismos de protección y 14 para especificar el segmento) y un campo de desplazamiento en el segmento de 32 bits. Con esta forma, el espacio de direcciones virtual que puede ver un usuario es de $2^{46} = 64$ Tbytes. Mientras que el espacio de direcciones físicas que se puede direccionar con 32 bits es de $2^{32} = 4$ Gbytes. Hay dos formas de protección asociadas con cada segmento:

1. *Nivel de privilegio*. Hay cuatro niveles de privilegio (del 0 al 3). Usualmente los niveles 0 y 1 corresponden al SO, el nivel 2 a algunos subsistemas y el nivel 3 a las aplicaciones.
2. *Atributos de acceso*. Para un segmento de datos indican si el permiso de acceso es de lectura/escritura o de sólo lectura; para un segmento de programa expresan si es de lectura/ejecución o de sólo lectura.

