Universidad Católica de Temuco

Escuela de Informática

Apuntes Sistemas de Operativos

Segunda Parte

Alejandro Mauricio Mellado Gatica
Magíster en Telecomunicaciones

Abril de 2003

Índice de contenido

Condiciones de concurso

En algunos sistemas operativos, los procesos que trabajan en conjunto frecuentemente comparten recursos como algún almacenamiento común que cada uno puede leer o escribir. El almacenamiento compartido puede ubicarse en la memoria principal o bien puede ser un archivo compartido. Las condiciones de concurso ocurren cuando dos o más procesos leen o escriben datos compartidos y el resultado final depende de cual se ejecuta en el momento preciso.

Secciones críticas

Para evitar las condiciones de concurso, de debe determinar de alguna manera de prohibir que un proceso lea o escriba los datos compartidos al mismo tiempo. Dicho de otra manera, lo que se necesita es exclusión mutua; la exclusión mutua es una manera de asegurar que si un proceso esta utilizando una variable o archivo es que los otros procesos un puedan acceder a estos recursos.

El problema de evitar condiciones de concurso también se puede formular en forma abstracta. Por parte del tiempo un proceso está ocupado realizando cálculos internos y otras tareas que no conducen a condiciones de concurso. Si embargo, a veces un proceso puede estar accediendo a memoria o archivos compartidos, o bien realizando otras tareas importantes que pueden llevar a competencias. Esa parte del programa desde se acceda a la memoria compartida se llama “sección crítica”. Si se pueden arreglar los acontecimientos para que dos procesos nunca estuvieran en sus secciones críticas al mismo tiempo, se podrían evitar las condiciones de concurso.

Se necesita que se cumplan cuatro condiciones para tener una solución adecuada:

1. Nunca dos procesos pueden encontrarse simultáneamente dentro de sus secciones críticas.
2. No se hacen suposiciones acerca de la velocidades relativas de los procesos o del número de CPUs.
3. Ningún proceso suspendido fuera de la sección crítica debe bloquear otros procesos.
4. Nunca un proceso debe querer entrar en forma arbitraria en su sección crítica.

Exclusión Mutua

Desactivación de Interrupciones

La solución más para lograr exclusión mutua, consiste en hacer que cada proceso desactive todas las interrupciones justo después de entrar en su sección crítica y las vuelva a activar antes de salir de ella. Con las interrupciones imposibilitadas, no pueden ocurrir interrupciones al reloj. (La CPU sólo cambia de un proceso a otro como resultado de interrupción de reloj u otras y con las interrupciones desactivadas la CPU no se cambiará a otro proceso).

Variables de cierre

Otra forma de crear exclusión mutua es a través de las variables de cierre. Considérese que se tiene una sola variable de cierre compartida que inicialmente vale 0. Cuando un proceso desea entrar en su sección critica, éste primero prueba el cerrojo. Si el cerrojo es 0, el proceso lo hace igual a 1 y entra en su región crítica. Si el cerrojo ya vale 1, el proceso simplemente espera hasta que se vuelva 0. Por lo tanto 0 significa que no hay ningún proceso en su sección crítica y 1 quiere decir que hay algún proceso en su sección critica.

Alternación Estricta

Un tercer método para el problema de la exclusión mutua se presenta en la siguiente rutina:

```
while (TRUE) {
        while (turno!=0)
                seccion_critica();
        turno=0;
        seccion_no_critica();
}
```

(a)

```
while (TRUE) {
        while(turno!=1)       /*esperar*/
                seccion_critica();
        turno=1;
        seccion_no_critica();
}
```

(b)

En esta solución propuesta, la variable entera turno, que inicialmente vale 0, lleva el control de los turnos de entrada en la sección crítica y examina o actualiza la memoria compartida. Inicialmente, el proceso (a) inspecciona el turno, y descubre que es 1 y entrar a su sección crítica el proceso (b) también determina que es 1 y por lo tanto se coloca en un ciclo cerrado donde prueba continuamente la variable turno para observar si se convierte a 0. A la continua comprobación de una variable que espera que aparezca algún valor se le

denomina “espera ocupada”. Por lo general se debe evitar, ya que desperdiciará tiempo de la CPU.

Cuando el proceso (a) sale de la sección crítica, hace un turno igual a 1 a fin de permitir que el proceso entre en su sección critica. Si el proceso (b) termina rápidamente su sección crítica, de modo que ambos procesos se encuentran en sus secciones no criticas, con turno igual a 0. Ahora el proceso (a) ejecutará todo su ciclo con rapidez, regresando a su sección no crítica con turno igual a 1. En este punto el proceso (a) termina su sección no crítica y regresa al ciclo de inicio. Por ahora, no se le permite entrar a su sección no critica, ya que turno vale 1 y el proceso (b) esta ocupado con su sección no crítica. Dicho de otra manera, tomar turnos no es adecuado cuando uno de los procesos es mucho más lento que el otro.

Solución de Peterson

Al combinar la idea de los turnos, con la de las variables de cierre y de advertencia, T. Dekker (Matemático Alemán) fue el primero en crear una solución por medio del software al problema de la exclusión mutua que no requiere alternación estricta. Esta solución fue muy complicada por lo que Peterson descubrió una manera mucho más simple de lograr exclusión mutua.
Código de la Solución de Peterson para lograr exclusión mutua.

```
#define FALSE 0
#define TRUE 1
#define N 2                                     /* Número de proceso */

int turno;                                      /* A Quién le toca ? */
int interes[N];                                 /* Todos los valores son inicialmente FALSE*/

ingresar_region(proceso)
int proceso;                                    /* Número del proceso */
{
        int otro;                               /* Número de otro proceso */

        otro = 1 – proceso;                     /* El opuesto del proceso */
        interes[proceso] = TRUE;                /* interés por ingresar a la sección crítica */
        turno = proceso;                        /*  Colocar señal */
        while ((turno == proceso) && (interes[otro]  == TRUE)); /* Posición nula */
}

salir_region(proceso)
int proceso;                                    /* Proceso que sale de la sección crítica */
{
        interested[proceso] = FALSE;            /* Indica la Salida de la sección crítica */
}
```

Sleep (Bloque) y Wakeup (Desbloqueo)

Algunas primitivas de comunicación entre procesos se bloquean en vez de gastar tiempo de CPU cuando no tienen permisos de entrar en sus secciones críticas. Una de las más simple es la pareja Sleep y Wakeup. Sleep es una llamada al sistema que hace que bloquee el solicitante, es decir, que se suspenda hasta que otro proceso lo desbloquee. La llamada Wakeup tiene un parámetro, una dirección de memoria que se usa para comparar llamadas Sleep con llamadas Wakeup.

Semáforos

En 1965, E.W. Dijkstra sugirió el uso de una variable entera para contar el número de desbloqueos guardados para uso futuro. En esta proposición se introdujo un nuevo tipo de variable, llamada semáforo. Un semáforo puede tener valor 0, lo que indicaba que no se habían guardado desbloqueos o algún valor positivo si quedaban pendientes uno o más bloqueos.

Dijkstra propuso contar con dos operaciones, down y up (generalizaciones de sleep y wakeup). La operación down y up con un semáforo verifica si el valor es mayor que 0. Si es así, disminuye el valor (es decir, utiliza un desbloqueo almacenado) o simplemente prosigue . Si el valor es 0, el proceso se bloquea. La verificación del valor, su alteración y posiblemente su bloqueo se realizan en una sola acción atómica indivisible. Se garantiza que una vez que se haya dado inicio a una operación del semáforo, ningún otro proceso puede accederlo hasta que se complete la operación.

La operación up incrementa el valor del semáforo direccionado. Si uno o más procesos se encuentran bloqueados en ese semáforo, y estos no pueden terminar una operación down anterior, el sistema elige uno de ellos al azar y le permite completar su operación down. Por lo tanto, después de una operación up con un semáforo con procesos bloqueados en el proceso bloqueado en el semáforo seguirá siendo 0, pero existirá un proceso bloqueado menos en él. La función de incrementar el semáforo y desbloquear un proceso también es indivisible.

Problema del productor y el consumidor utilizando semáforos.

```
#define N 100                    /* Número de ranuras del buffer */
typedef int semáforo;            /* tipo semáforo como entero */
semaforo exclusion=1;            /* controla el acceso a la sección crítica */
semaforo vacio=N;                /* cuenta las ranuras vacías del buffer */
semáforo lleno=0;                /* cuenta las ranuras llenas del buffer */
```

```
productor()
{
        int item;

        while (TRUE) {                          /* ciclo infinito */
                procedimiento_item(&item);      /* generar algo para colocarlo en el buffer */
                down(vacio);                    /* disminuir el conteo vacío */
                down(exclusion);                        /* meter en la sección critica */
                ingrese_item(item);             /* colocar el nuevo elemento en el buffer */
                up(exclusion);                          /* salir de la sección crítica */
                up(lleno);                      /* incrementar el conteo de ranuras llenas */
        }
}

consumidor()
{
        int item;

        while(TRUE) {                           /* ciclo infinito */
                down(lleno);                    /* disminuir el conteo lleno */
                down(exclusion);                /* meter en la sección crítica */
                remover_item(&item);            /* tomar el elemento del buffer */
                up(exclusion);                  /* salir de la sección crítica */
                up(vacio);                      /* incrementar el conteo de ranuras vacías */
                consume_item(item);             /* hacer algo con el elemento */
        }
}
```

Esta solución emplea tres semáforos, uno llamado lleno (full) para contar el número de ranuras que están llenas, uno denominado vacío (empty) para contar el número d ranuras que están vacías y uno conocido como exclusión (mutex) para asegurar que el productor y el consumidor no accedan al buffer al mismo tiempo. La variable lleno inicialmente está en 0, la variable vacío es igual al número de ranuras en le buffer y exclusión es 1. Los semáforos se inician en 1 y son utilizados por dos a más procesos para garantizar que sólo uno de ellos pueda entrar en su sección crítica al mismo tiempo se denominan semáforos binarios. Si cada proceso realiza una operación down justo antes de entrar en su sección crítica y un up justo después de salir de ella, se asegura que exista exclusión mutua.

Administración de Memoria

La ley de Parkinson nos dice que los “los programas se desarrollan para ocupar toda la memoria disponible en ellos” [1].

La parte del Sistema Operativo que administra la memoria se llama “ Administrador de Memoria”. Su labor consiste en llevar un registro de las zonas de memoria que se estén utilizando y aquellas que no, con el fin de asignar espacio de memoria a los procesos cuando estos la necesiten y liberar el espacio en memoria cuando ya no se necesite, así como también administrar el intercambio de entre la memoria principal y el disco, en el caso que la memoria principal no pueda contener a todos los procesos.

Administración de memoria sin paginación o intercambio

Los sistemas de administración de la memoria se pueden clasificar en dos tipos: los que desplazan los procesos de la memoria principal al disco y viceversa durante la ejecución (intercambio y paginación) y los que no.

La mono programación sin intercambio o paginación

El esquema más sencillo de administración de memoria es aquel en que sólo se tiene un proceso en memoria a cada instante. En este caso puede cargarse toda la memoria con un programa del disco en cinta.

La técnica usual de los microcomputadores simples se muestra en la figura 1. La memoria se divide entre el Sistema Operativo y el proceso de un solo usuario. EL Sistema Operativo, podría estar en la parte inferior de la memoria RAM, como se muestra en la figura 1(a) o en la parte superior de la memoria como la figura 1(b), o los controladores de dispositivos podrían estar en la parte superior de la memoria ROM y el resto del Sistema Operativo en la parte inferior de la memoria, como se muestra en la figura 3(c). Los PC x86 por ejemplo, utilizan el modelo de la figura 3(c), con los controladores de dispositivo en la ROM, en un bloque de 8k en la parte superior del espacio de direcciones de 1M. El programa en ROM se llama BIOS (Basic Input Output System, Sistema Básico de Entrada y Salida) con este tipo de organización se puede ejecutar un solo proceso a la vez.

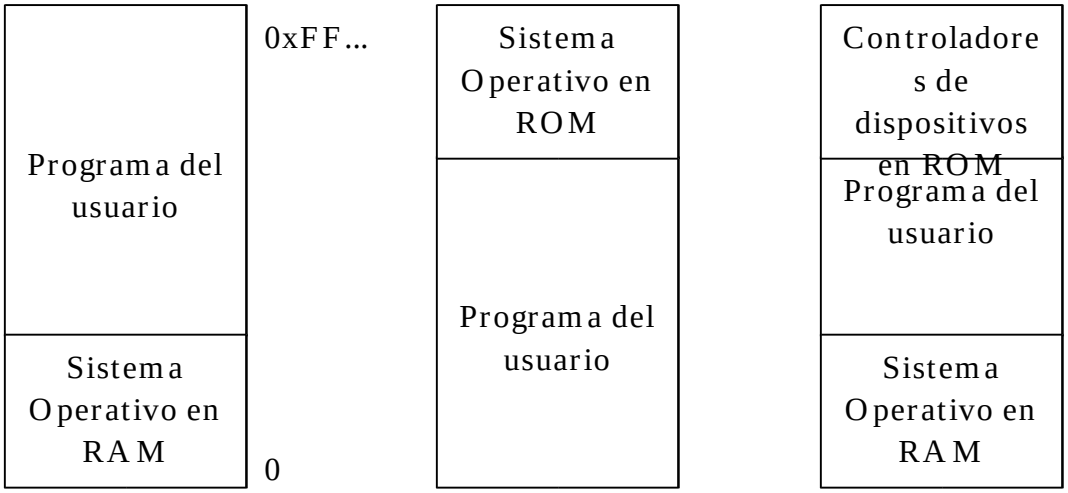


Figura 1: Tres formas de organización de la memoria, con un sistema operativo y un proceso del usuario.

El uso de la memoria de la multiprogramación

Aunque la monoprogramación se utiliza en ciertas ocasiones en los computadores pequeños, las grandes computadores con muchos usuarios casi no usan la monoprogramación . Los grandes computadores proporcionan un servicio interactivo a muchos usuarios al mismo tiempo, lo que requiere de la capacidad de tener más de un proceso en la memoria a la vez, con el fin de obtener un desempeño razonable.

Modelo de multiprogramación

El uso de la CPU se puede mejorar mediante la multiprogramación . En teoría si el proceso promedio hace cálculos sólo durante el 20% del tiempo que permanece en la memoria y tiene 5 procesos en la memoria la mismo tiempo, la CPU debería estar ocupada todo el tiempo. Sin embargo este es un optimismo irreal, puesto que los 5 procesos nunca esperan entrada y salida el mismo tiempo .

Un mejor modelo consiste en analizar el uso de la CPU desde un punto de vista probabilístico. Supongamos que un proceso ocupa una fracción p de su tiempo en el estado de espera de E/S. Si n procesos se encuentran en la memoria al mismo tiempo, la probabilidad de que los n procesos esperen por E/S sería p^n. El uso de la CPU estaría dado entonces por la formula

$$\text{Uso de la CPU} = 1 - p^n$$

Multiprogramación con particiones fijas

La forma más sencilla de dividir la memoria es en n partes (que podrían ser de distintos tamaños). “Esta partición podría hacerla el operador en forma manual, al iniciar un sesión con la máquina”(ahora esto lo hace en forma automática el sistema operativo).

Cuando llega un trabajo, se le puede colocar en la cola de entrada de la parte de tamaña más pequeño de forma que lo pueda contener. Puesto que las particiones están fijas en este esquema, cualquier espacio que no sea utilizado por una tarea se pierde. En la figura 3(a) se ve el esquema de particiones fijas y colas de entrada independientes.

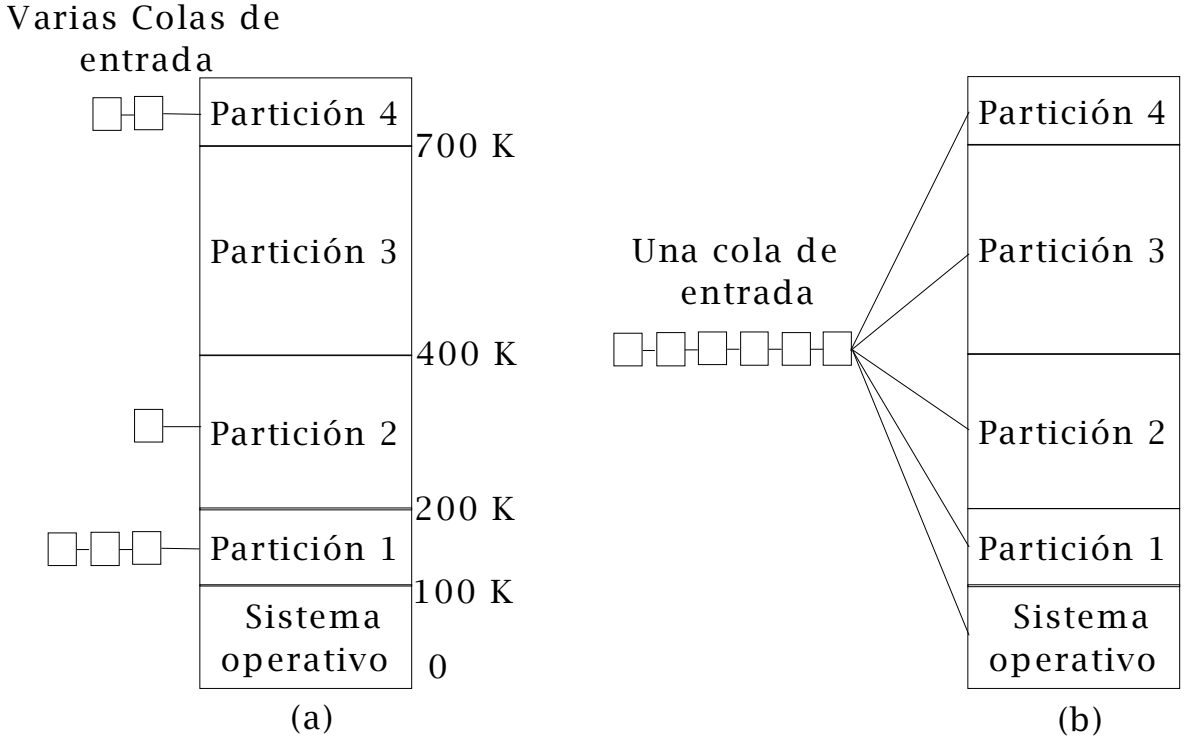


Figura 3. (a) Particiones fijas de memoria con colas de entrada independientes para cada partición. (b) Particiones fijas de la memoria, con una única cola de entrada.

La desventaja del ordenamiento de las tareas que llegan a la memoria en colas independientes es evidente cuando la cola de una partición grande está vacía pero la cola de una partición pequeña esta completamente ocupada, como este el caso de la particiones 1 y 4 de la figura 3(a). Otro tipo de organización es el que mantiene una sola cola como en la figura 3(b). Cada vez que se libere una partición, se podría cargar en ella la tarea más cercana al frente de la cola que ajuste a dicha partición. Puesto que no es deseable que se desperdicie una partición de gran tamaño con una tarea pequeña, otra estrategia consiste en

buscar en toda la cola de entrada el trabajo más grande que se ajuste a la partición recién liberada. Obsérvese que este último algoritmo discrimina a las tareas pequeñas, negándoles la importancia suficiente como para disponer de toda una partición, mientras que lo recomendable es que se les dé a las tareas pequeñas el mejor servicio y no el peor.

Una forma de salir del problema es tener siempre una pequeña partición por ahí. Tal partición permitiría la ejecución de las tareas pequeñas sin tener que asignarle una partición grande.

Otro punto de vista es obedecer como regla que un trabajo elegible para su ejecución no sea excluido más de k veces. Cada vez que se le excluya, obtiene un punto. Cuando adquiera k puntos, ya no podrá ser excluido de nuevo.

Intercambio

En un sistema por lotes, la organización de la memoria en particiones fijas es sencilla y eficaz. Mientras que se mantengan dentro de la memoria la cantidad suficiente de programas como para que la CPU esté ocupada todo el tiempo. Sin embargo, la situación es distinta con el tiempo compartido: por lo general, existen más usuarios de los que puede albergar la memoria, por lo que es necesario mantener el exceso de los procesos en disco. Por supuesto, para ejecutar esos procesos, deben ser trasladados a la memoria principal. El traslado de los procesos de la memoria principal al disco y viceversa se llama intercambio.

Multiprogramación con particiones Variables

En principio, las particiones fijas se podrían utilizar par un sistema con intercambio. Cada vez que se bloquea un proceso, se podría trasladar el disco y llevar otros proceso a la partición ocupada por el primero. En la practica, las particiones fijas no son muy atractivas si se dispone de poca memoria, puesto que la mayor parte de la ésta se desperdicia con programa menores que sus particiones. En vez de esto, se utiliza otro algoritmo de administración de la memoria conocido como “particiones variables”.

Al utilizar las particiones variables, el número de y tamaño de los procesos en la memoria varía en forma dinámica a medida del uso. La figura 4 muestra el funcionamiento de las particiones fijas. AL principio, sólo se encuentra el proceso A en la memoria. Después se crean los procesos B y C o se intercambian con el disco. En la figura 4(d), A concluye o se pasa al disco. Luego D entra y B sale. Por último, E entra.

La diferencia principal entre las particiones fijas de la figura 3 y las particiones variables de la figura 4 es que el número, posición y tamaño de las particiones varía en forma dinámica en el segundo caso al crearse o intercambiarse más procesos, mientras que en el primer caso estas características son fijas. El hecho de no estar sujeto a un número fijo de particiones que pudieran ser muy grandes o demasiado pequeñas mejora el uso de la

memoria pero también hace más compleja la asignación y re−asignación de memoria, así como también complica el mantener un registro de esto.

Es posible combinar todos los huecos en uno grande, si se mueven todos los procesos hacia la parte interior, mientras sea posible. Esta técnica se conoce como *compactación de memoria*. Por lo general, no se lleva a cabo, porque consume mucho tiempo de CPU. Por ejemplo una máquina de 1M que pueda copiar un byte por microsegundo (1 megabyte/seg) tardaría 1 segundo en compactar toda la memoria. Sin embargo la antigua CDC Cybers se levaba a cabo la compactación, puesto que tenía un hardware especial y podía compactar a razón de 40 magebytes/seg.

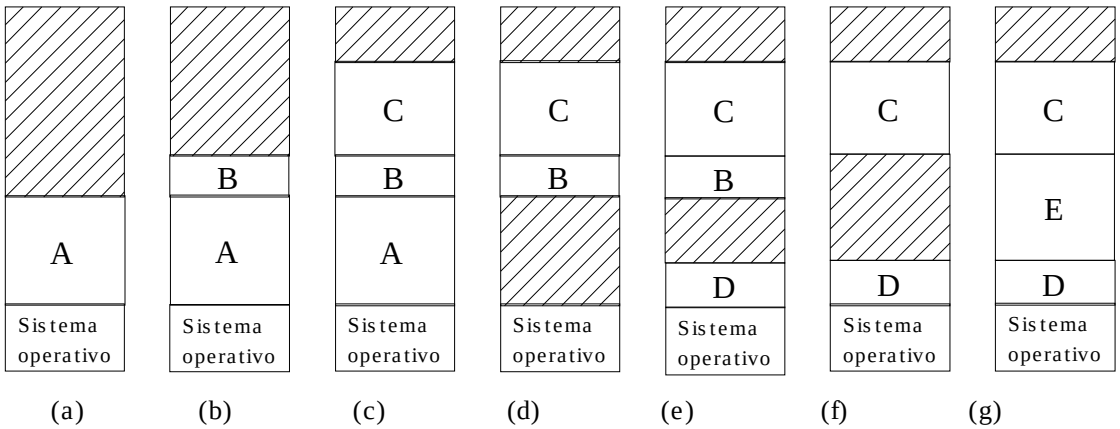


Figura 4. la asignación de memoria cambia cuando el proceso entra o sale de la memoria.

Un aspecto importante es la cantidad de memoria por asignar a un proceso recién creado o intercambiado. Si los procesos se crean con un tamaño fijo invariante, entonces la asignación es muy sencilla, ya que se asigna sólo lo que se necesite.

Si los segmentos de datos de los procesos pueden crecer, como es el caso de la asignación dinámica de memoria a partir de una pila, como es el caso en muchos lenguajes de programación, aparece un problema cuando un proceso intenta crecer. Si existe un hueco adyacente a otro proceso, el proceso de crecimiento deberá ser desplazado a un hueco de la memoria lo suficientemente grande; o bien, habrá que intercambiar uno o más procesos para crear un hueco grande. Si un proceso no puede crecer dentro de la memoria y el área de intercambio está llena, el proceso deberá esperar o ser aniquilado.

Memoria Virtual

La idea de la memoria virtual se basa en que el tamaño combinado del programa, los datos y la pila pueden exceder la cantidad de memoria física disponible para él. El sistema operativo mantiene aquellas partes del programa que se utilicen en cada momento en memoria principal y el resto permanece en el disco duro. Por ejemplo, un programa de 1 M se pude ejecutar en una máquina de 256 K, si se eligen con cuidado los 256 K del programa que deben permanecer en memoria en cada instante; las partes se intercambian entre el disco y la memoria según sea necesario.

La memoria virtual también sirve en un sistema de multiprogramación. Por ejemplo, ocho programas de 1 M se pueden asignar en una partición de 256 K de una memoria de 2 M, como si cada programa operara teniendo su propia máquina particular de 256 K. De hecho, la memoria virtual y la multiprogramación se complementan. Mientras que un programa espera a que parte de él sea intercambiado, espera una E/S y no puede ejecutarse, de forma que la CPU puede asignarse a otro proceso.

Paginación

La mayoría de los sistemas con memoria virtual utilizan una técnica llamada paginación. En cualquier computador, existe un conjunto de direcciones de memoria que pueden producir los programas. Cuando un programa utiliza una intrusión como por ejemplo

MOVE REG, 1000

Copia el contenido de la dirección de la memoria 1000 a REG (o viceversa, según el computador). Las direcciones se pueden generar mediante índices, registros de base, registros de segmentos y otras formas.

Estas direcciones generadas por los programas se llaman direcciones virtuales y conforman el hueco de direcciones virtuales. En los computadores que no tienen memoria virtual, la dirección virtual se coloca en forma directa dentro del bus de la memoria, lo cual hace que se pueda leer o escribir en la palabra de la memoria física que tenga la misma dirección. AL utilizar la memoria virtual, la direcciones virtuales no pasan en forma directa el bus de memoria, sino que van al a unidad de administración de memoria (MMU), un chip o un conjunto de chips que asocian las direcciones virtuales con la memoria física, como se muestra en la figura 5.

Como ejemplo del funcionamiento de esta asociación, veamos la figura 6, en donde tenemos un computador que puede generar direcciones de 16 bits, que van desde 0 a 640K. Estas son las direcciones virtuales. Sin embargo, este computador sólo tiene 32 K de memoria física, de forma que aunque se puedan escribir programas de 64 K, éstos no se pueden cargar en la memoria en su totalidad para ser ejecutados. Sin embargo, una copia

completa de la imagen del núcleo del programa, de hasta 64K, debe estar presente en el disco, de forma que se puedan utilizar sus partes en caso necesario.

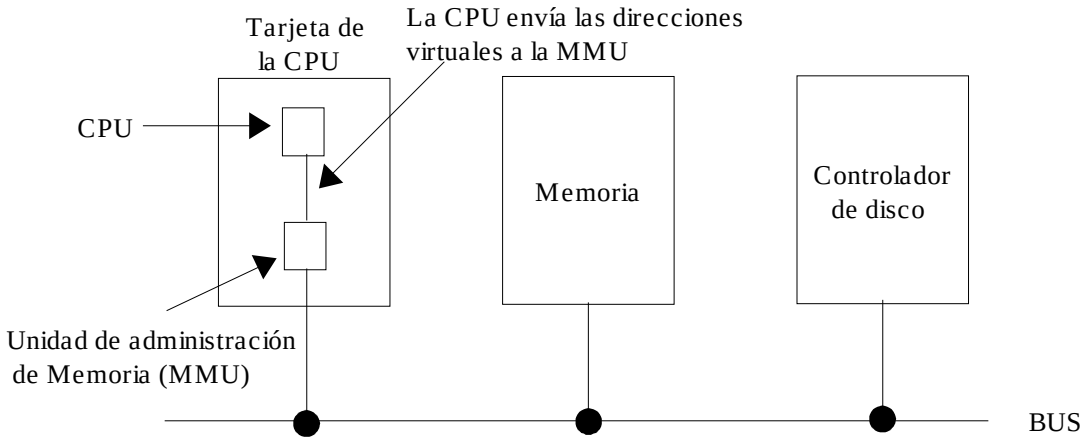


Figura 5. Posición y función de las MMU.

El hueco de direcciones virtuales se divide en unidades llamadas páginas. Las unidades correspondientes en la memoria física se llaman marco para página. Las páginas y los marcos tienen siempre el mismo tamaño. En este ejemplo, son de 4K, pero los tamaños de 512 bytes a 8 Kbytes también son de uso común. Con 64 K de hueco de direcciones virtuales y 32 de memoria física, tenemos 16 páginas virtuales y 8 marcos. Las transferencias entre la memoria y el disco son siempre por unidades de página.

Por ejemplo, cuando el programa intenta tener acceso a la dirección 0, mediante la instrucción

MOVE REG, 0

La dirección virtual 0 se envía a la MMU. La MMU ve que esta dirección virtual cae dentro de la página 0 (0 a 4095), la cual, de acuerdo con su regla de correspondencia, esta en el marco 2 (8192 a 12287). Transforma entonces la dirección en 8192 y manda la dirección 8192 al bus. EL tablero de la memoria no sabe acerca de la MMU y solo ve una solicitud de lectura y escritura de la dirección 8192, a la que da paso.

Así, la MMU ha asociado todas las direcciones virtuales entre 0 y 4095 con las direcciones físicas 8192 a 12287.

En forma analógica, instrucción

MOVE REG, 8192

Se transforma en

MOVE REG, 24576

Puesto que la dirección virtual 8192 está en la página virtual 2 y esta página esta asociada a un marco físico de 6 (direcciones físicas 24576 a 28671). Como tercer ejemplo, la dirección virtual 21500 está a 20 bytes del inicio de la página virtual 5 (direcciones virtuales 20480 a 24575) y está asociada a la dirección física12288+20=12308,

En si misma, esta capacidad para asociar las 16 páginas virtuales con cualquiera de los ocho marcos para página al determinar una regla de correspondencia adecuada de las MMU no resuelve el problema de que el hueco de direcciones virtuales es más grande que la memoria física.

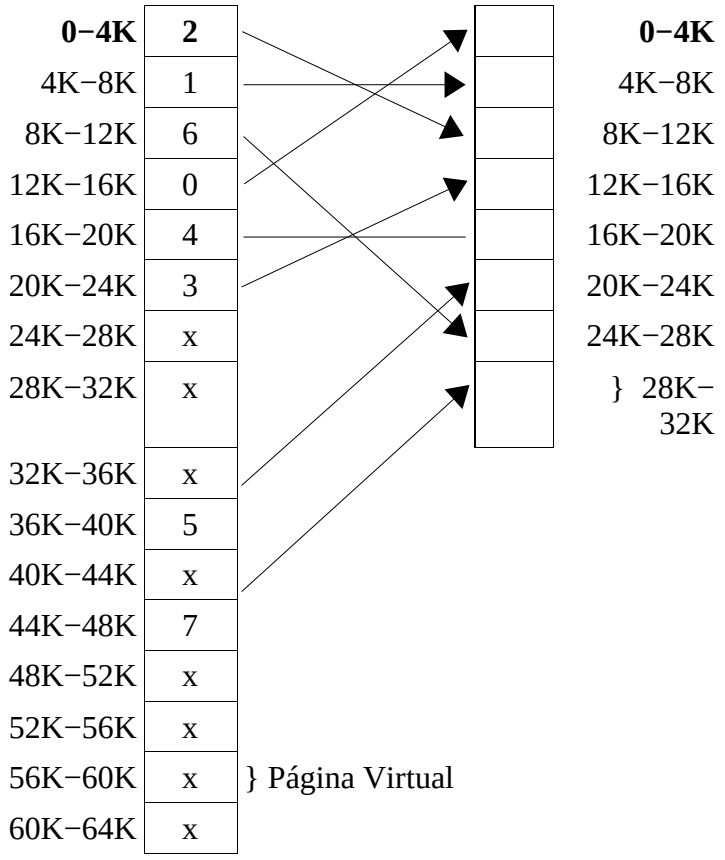


Figure 6. La relación entre las direcciones virtuales y las direcciones en la memoria física está dada en la tabla de páginas.

Puesto que solo tenemos ocho marcos para página, sólo ocho de las páginas virtuales d la figura 6 se asocian a la memoria física. Las demás, que aparecen con una X en la figura, no quedan asociadas. En el hardware real, un *bit presente o ausente* de cada entrada mantiene un registro de si la página está asociada o no.

¿Qué pasa si el programa intenta utilizar una página no asociada, por ejemplo, con la instrucción

MOVE REG, 32780

que está en el byte 12 dentro de la página virtual 8 (la cual comienza en 32768)?. La MMU observa que la página no está asociada (lo cual se indica con una X en la figura) y hace que la CPU utilice un señalamiento hacia el sistema operativo. Este señalamiento (trap) se llama un fallo de página. El sistema operativo elige un marco para página de poco uso y escribe su contenido en el nuevo disco. Después busca la página de referencia al marco liberado, modifica la asociación y reinicia a instrucción señalada.

Por ejemplo, si el sistema operativo decidió liberar el marco 1, cargará la página virtual 8 en la dirección física 4K y hará dos modificaciones al mapa de las MMU. En primer lugar indica que la entrada de la página virtual 1 no está asociada, para señalar los futuros accesos de las direcciones virtuales entre 4K y 8K. Entonces remplazaría la cruz en la entrada de la página virtual 8 por 1, de forma que cuando la instrucción señalada se vuelva a ejecutar, asocie la dirección virtual 32780 con la dirección física.

Entrada y Salida (Principios de hardware)

Una de las funciones principales de un Sistema Operativo consiste en controlar todos los dispositivos de entrada y salida de la computadora. Este debe emitir comandos a los dispositivos, capturar interrupciones y manejar errores. También debe proporcionar una interfaz entre los dispositivos y el resto del sistema que sea simple y fácil de utilizar. Hasta donde sea posible, la interfaz debe ser la misma para todos los dispositivos, creando una independencia del dispositivo. El código de entrada y salida (E/S) representa un porción significativa del sistema operativo en su totalidad.

Principios de Hardware de E/S

“El Hardware es percibido de diferentes formas dependiendo de la persona”. Los Ingenieros eléctricos los perciben en términos de chips, alambres, fuentes de energía, motores y todas las otras componentes físicas que lo conforman. Los programadores en cambio, lo perciben como la interfaz que se presenta al software, o sea, los comandos que hardware acepta, las funciones que lleva a cabo y los errores que puede informar.

Dispositivos de E/S

Los dispositivos de E/S pueden dividirse aproximadamente en dos categorías: Dispositivos de Bloque y dispositivos de caracter.

Un *dispositivo de Bloque* es aquel que almacena información en bloques de tamaño fijo, cada uno con su dirección. Los tamaños comunes de los bloques van de 128 bytes a 1024 bytes. La propiedad esencial de un dispositivo de bloque es qué es posible leer y escribir cada bloque en forma independiente de los demás. En otras palabras el programa, en cualquier momento, puede leer o escribir en cualquiera de los bloques. Por lo general correspondes a este tipo de dispositivos los discos duros y/o medios de almacenamiento.

El otro dispositivo de E/S es el *dispositivo de caracter*. Un dispositivo de caracter entrega o acepta un flujo de caracteres, sin importar de cuál estructura de bloque se trate. Esto no es direccionable y no se tienen operaciones de localización. Los terminales (TTY), las impresoras seriales, las interfaces de redes, los ratones y todos los dispositivos que accedan a bloques de tamaño fijo pueden ser considerados dispositivos de caracter.

Controlador de dispositivos

Las unidades de E/S por lo general constan de un componente mecánica y una electrónica. A menudo es posible separar las dos porciones para ofrecer un diseño más modular y general. La componente electrónica se denomina *controlador de dispositivo o adaptador*. En los computadores personales por lo general son las tarjetas que dan funcionalidades adicional a computador y que son insertados en los distintos tipos de “buses o slots de expansión”.

Debe hacerse una distinción entre el controlador de dispositivo y el dispositivo porque cada sistema operativo casi siempre interactúa con el controlador, no con el dispositivo. Casi todos los computadores personales y los servidores (microcomputadores) utilizan el modelo de BUS para establecer comunicación entre la CPU y los controladores de dispositivos (ver figura 7). Los grandes computadores con frecuencia usan un modelo diferente, con múltiples buses y computadores especializados que se denominan canales de E/S que toman algo de carga de CPU, actualmente esta característica ha sido incorporada a las últimas generaciones de computadores personales.

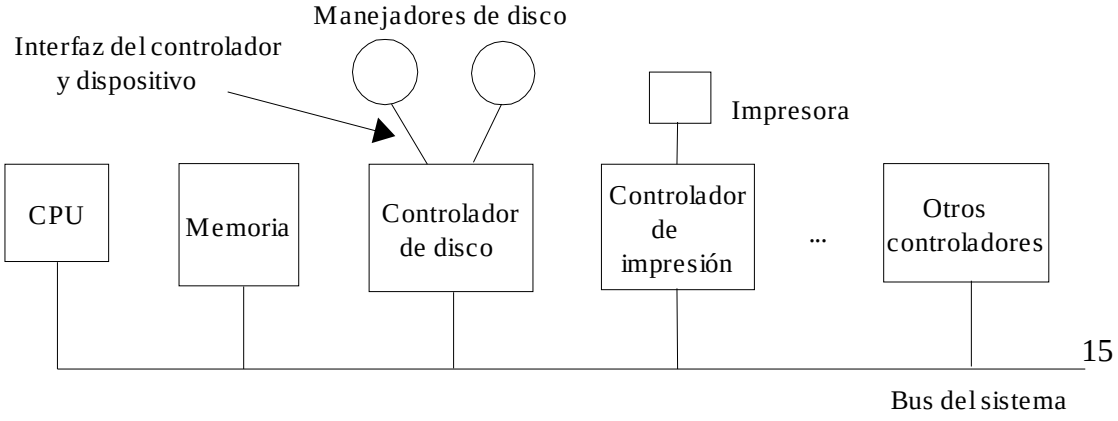


Figura 7. Modelo para conectar la CPU, la memoria, controladores y dispositivos de E/S.

El trabajo del controlador de consiste en convertir el flujo de bits en serie, en un bloque de bytes y realizar cualquier corrección de errores que necesite. El bloque de bytes primero se ensambla comúnmente, bit por bit, en un buffer en el interior del controlador. Después que se ha verificado su suma de comprobación y que el bloque se ha declarado libre de error, puede copiarse entonces en la memoria principal.

Cada controlador tiene algunos registros que se utilizan para establecer comunicación con la CPU. En algunos computadores, estos registros son parte del espacio de dirección de la memoria principal. Los que utilizan un espacio de dirección especial para E/S, donde cada controlador se la asigna un porción determinada de ella.

Ejemplo de controladores, sus direcciones de E/S y sus vectores de interrupción (IRQ):

Controlador de E/S	Dirección de E/S	Vector de Interrupción
Reloj	0x040	0
COM1	0x3F8	4
COM2	0x2F8	3
DISCO DURO	0x3F6	14
LPT1	0x378	7

Acceso Directo a Memoria (DMA)

Muchos controladores, en especial los dispositivos de bloque, poseen acceso directo a memoria. Para explicar como funciona el Acceso Directo a Memoria (DMA), se de describirá primero como ocurren las lecturas a disco cuando no se utiliza DMA. Primero el controlador lee uno o más sectores (bloque) de la unidad en serie, bit por bit, hasta que todo el bloque este en el buffer interno del computador. Después realiza el cálculo de la suma de comprobación para verificar que hayan ocurrido errores en la lectura. Después el controlador produce una interrupción. Cuando el sistema operativo inicia su ejecución, este puede leer un bloque o una palabra a la vez, ejecutando un ciclo, donde en cada iteración se lee un byte o una palabra de un registro del dispositivo controlador y se almacena en la memoria. Un ciclo programado en la CPU para leer uno por uno los bytes del controlador gasta tiempo de dicha unidad. El DMA se inventó para liberar a la CPU de este trabajo de bajo nivel. Cuando se utiliza, esta unidad da al controlador dos elementos de información,

además de la dirección en disco del bloque: la dirección de la memoria a donde se dirige el bloque y el número de bytes por transferir como se muestra en la figura 8.

Después de que el controlador ha leído todo el bloque del dispositivo en su buffer y verificado la suma de comprobación, éste copia el primer byte o palabra en la memoria central en la dirección especificada por la dirección de memoria DMA. Después incrementa la dirección DMA y determina el conteo DMA por el número de bytes que acaba de transferirse. Este proceso se repite hasta que el conteo DMA se vuelve cero, y en ese instante el controlador da origen a una interrupción. Cuando un sistema operativo inicia su proceso, no tiene que copiar el bloque en la memoria: este ya se encuentra allí.

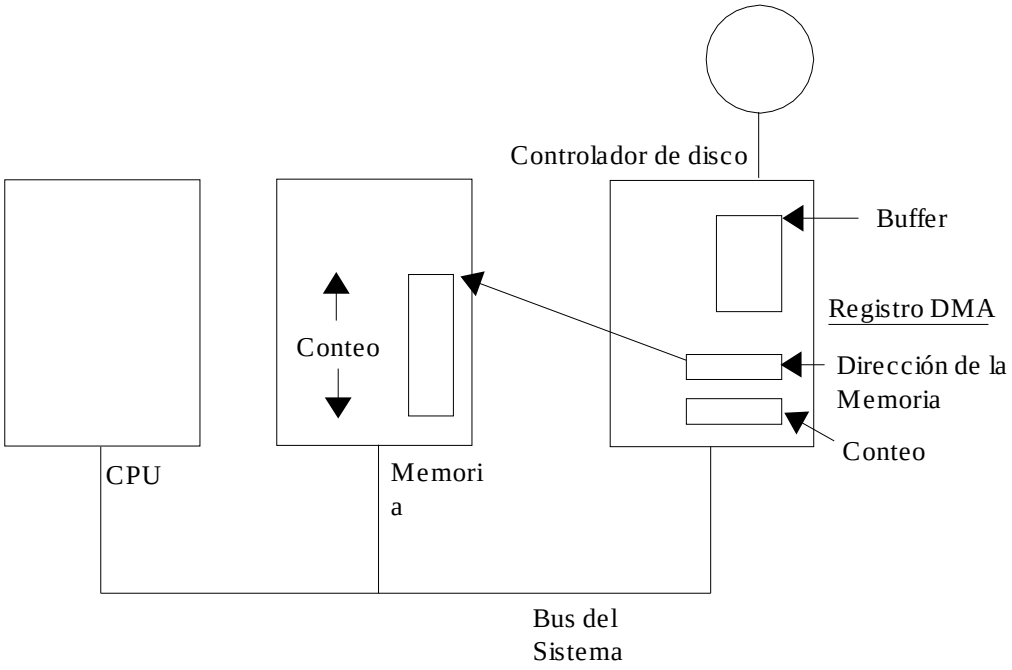


Figura 8, La transferencia DMA es efectuada completamente por el controlado

Principios del Software de E/S

Un concepto importante en el desempeño del software de E/S es la “independencia del dispositivo”. Debe se posible escribir programas que puedan utilizarse con archivos en un disco flexible o un disco duro, sin tener que modificar los programas para cada tipo de dispositivo. De hecho, debe ser posible desplazar el programa sin tener que recompilarlo. Corresponde al sistema operativo hacerse cargo de los problemas ocasionados por el hecho de que estos dispositivos en realidad son diferentes y requieren de manejadores de dispositivos muy diferentes.

Otro aspecto importante de E/S es el manejo de los errores. En términos de generales los errores deben mantenerse lo más apegados al hardware que sea posible. Si el controlador de dispositivo descubre un error de lectura, debe intentar corregirlo si es posible hacerlo. Si no puede hacerlo, entonces el manejador de dispositivo debe hacerse cargo de él, quizá simplemente intentando volver a leer el bloque. Muchos errores son transitorios, como los errores de lectura ocasionados por partículas de polvo en la cabeza de lectura de un disco duro, y desaparecerán, si la operación se repite. Sólo si los estratos inferiores no pueden resolver el problema se debe informar de él a los superiores.

Sistemas tolerantes a fallos

El RAID

La idea básica de RAID (Redundance Arrays of Inexpensive Disk) es combinar múltiples dispositivos de discos económicos en un arreglo de disco que ofrece buen rendimiento. Adicionalmente, este arreglo de dispositivos aparece frente al computador como una simple unidad lógica o dispositivo.

Se ha definido 5 tipos de arquitecturas de arreglo. Desde el RAID 0 hasta el RAID 5, cada uno de los arreglos provee tolerancias a fallas y cada uno ofrece diferentes combinaciones en características y rendimiento. Además para esas 5 arquitecturas de arreglos redundantes, esto ha llegado a ser una referencia popular arreglos de discos sin redundancia como es el caso del RAID 0.

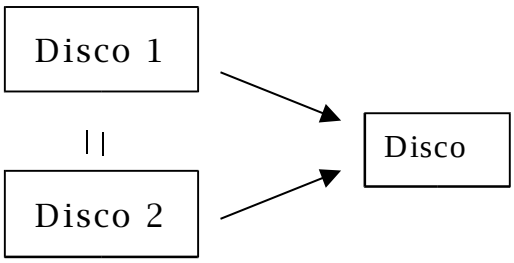
Diferentes niveles de RAID

RAID 0

El nivel RAID 0 no es redundante, por lo tanto no se ajusta exactamente a la definición del acrónimo RAID. En el nivel 0, los datos son divididos en los dispositivos, resultando una más alta transferencia de datos. Como los datos son almacenados sin redundancia, el rendimiento es muy bueno, pero la falla de cualquiera de sus discos generará una pedida de datos. Este nivel es comúnmente denominado volumen (striping).

RAID 1

El nivel RAID 1 provee redundancia por escritura en dos o más disco. El rendimiento del arreglo de nivel 1 tiende a ser más rápido en las lecturas y más lento en las escrituras comprado con un simple dispositivo, pero si uno de los dos falla, no se perderán los datos. Esta es un buen nivel de entrada redundante del sistema, como solamente se requieren dos dispositivos, uno de los dispositivos es usado para recibir la los datos quedando una duplicado o copia exacta del otro dispositivo, en este caso el costo por megabyte es alto (½). Este nivel es generalmente llamado espejo (mirroring).



RAID 2

El nivel RAID , que uso el código de corrección de errores de Hamming, esta proyectado para uso con dispositivo que no tienen detección de errores empotrado en su hadware. Todos los dispositivos SCSI tienen empotrado en su hardware mecanismos para la detección de errores, así que este nivel es poco frecuente cuando se usan dispositivos SCSI.

RAID 3

El nivel RAID 3 divide los datos en un byte en distinto dispositivos, con almacenamiento de paridad en un dispositivo. Esto es de otro modo similar al nivel 4. Dividir el nivel de bytes que requiere el hardware para su uso eficiente.

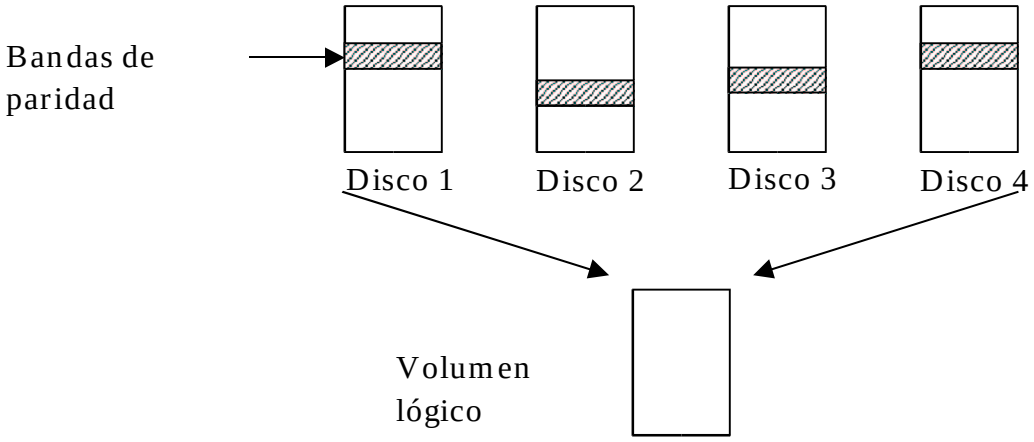
RAID 4

El nivel RAID 4 divide los datos en un nivel de bloque en distintos dispositivos, con almacenamiento de paridad en un dispositivo. La información de paridad permite recuperar la falla de cualquier dispositivo simple. El rendimiento del arreglo de nivel 4 es muy bueno para lectura. La escritura, sin embargo, requiere que los datos de paridad sean actualizados a cada momento. Como solamente un dispositivo en el arreglo almacena los datos redundante, el costo por megabyte del nivel 4 puede ser bastante bajo.

RAID 5

El nivel RAID 5 es similar al nivel 4, pero la distribución de la paridad se reparte entre los dispositivos. Esto puede acelerar la escritura de pequeños bloques de bytes en sistemas

multiprocesados, así la paridad de disco no ha llegado a ser un cuello de botella. Ya que la paridad de datos debe ser saltada sobre cada dispositivo durante las lecturas, el rendimiento de lectura tiende a ser considerablemente más bajo que en el arreglo de nivel 4.
El costo por megabyte es el mismo del nivel 4.



Los Clusters de servidores

Definición

Un cluster de servidores es un grupo de servidores independientes administrados por un sistema único para más alta disponibilidad, administrabilidad y escalabilidad.

(*Definición del sitio de Microsoft)*

Greg Pfister, define Cluster como "un tipo de sistema distribuido o paralelo que esta compuesto por un colección de computadores totalmente interconectados, y que están usado como un simple recurso de computación unificado". Los Clusters ha sido concebidos, formalmente o informalmente, de muchos tipos de sistemas.

Referencias

[1] Sistemas Operativos Modernos, Andrew S. Tanenbaum, Pearson Educación, 1993.

[2] http://www.uni−mainz.de/~neuffer/scsi/what_is_raid.html