UNIVERSIDAD LASALLISTA BENAVENTE

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

Con estudios incorporados a la

Universidad Nacional Autónoma de México

CLAVE: 8793-16

"SISTEMAS MULTIPROCESADORES"

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERA EN COMPUTACIÓN

P R E S E N T A :

ANA MARÍA CAMPOS MATA

ASESOR: ING. NÓE DE JESÚS VELA AGUIRRE

CELAYA, GUANAJUATO **MARZO DE 2003**

A

AGRADECIMIENTOS

A Dios, por otorgarme caminar por la vida y por estar siempre a mi lado. Le agradezco por permitirle a mis padres haberles proporcionado los medios para que yo pudiera recibir una excelente educación y el tener las fuerzas necesarias para lograr el éxito en mi etapa de estudiantil.

A mis Padres, por apoyarme en mis logros y flaquezas, por proporcionarme las herramientas necesarias para llegar al fin de una etapa muy bella el ser estudiante; por el sacrificio para brindarme los recursos, porque siempre han procurado mi bien y sin ellos hubiera sido imposible la realización de este sueño gracias a ellos fue posible culminar este sueño.

A mis Hermanos, les agradezco la paciencia que tuvieron en los sacrificios de mis padres para que ellos me sacaran adelante, por el apoyo diario a seguir estudiando y preparando.

A mis Maestros, por haber compartido su sabiduría, paciencia y experiencia para que yo forje mi futuro con sus enseñanzas.

ÍNDICE

CAPITULO 4. PROCESOS, CUESTIONES DE DISEÑO E IMPLEMENTACIÓN DEL SISTEMA OPERATIVO

CAPITULO 5. INTRODUCCIÓN A LA PROGRAMACIÓN PARALELA

CAPITULO 6. SINCRONIZACIÓN EN MULTIPROCESADORES

CAPÍTULO 7. SISTEMAS MULTIPROCESADORES COMERCIALES

INTRODUCCIÓN

El tema de Sistemas Multiprocesadores han tenido un gran impacto debido a la productividad que presenta el sistema y la ganancia de velocidad de varios procesadores de alta velocidad en paralelo que lo conforman y van dirigidos a aplicaciones que requieren de un alto desempeño en el procesamiento de información. Por tal motivo se eligió este tema con el fin de tener un conocimiento pleno del mismo, las plataformas en las cuales se pueden tener.

Se inicia conociendo que es un sistema multiprocesador, la clasificación, características y ventajas de los sistemas multiprocesadores. Continuado con la descripción de sus estructuras de interconexión para construir un multiprocesador comprendiendo las arquitecturas básicas (simples), conociendo los tipos básicos de sistemas multiprocesadores explicando detalladamente las exigencias funcionales y los métodos de diseño de los sistemas operativos multiprocesadores se discuten las facilidades y técnicas que emplean los sistemas operativos para gestionar procesos y hebras de ejecución, muestra la ganancia de velocidad de aplicaciones en multiprocesadores y contiene un tratamiento detallado del soporte hardware para la sincronización entre procesadores en sistemas multiprocesadores. Presenta ejemplos de sistemas multiprocesadores comerciales disponibles en el mercado.

F

CAPITULO UNO

MULTIPROCESADORES

1.1 ¿QUÉ ES UN SISTEMA MULTIPROCESADOR?

Un sistema multiprocesador es una interconexión de dos o más CPU con un equipo de memoria y entrada-salida.[1] Por lo tanto, implica la existencia de múltiples CPU, por lo general siempre habrá uno o más procesadores de entrada-salida (IOP).

Un sistema multiprocesador lo controla un sistema operativo que proporciona interacción entre los procesadores y todos los componentes del sistema cooperan en la solución de un problema. Sin embargo, algunas computadoras de gran tamaño incluyen dos o más CPU en su sistema general; con el surgimiento del microprocesador produce una gran motivación por los sistemas multiprocesadores, por la razón de que los microprocesadores ocupan muy poco espacio físico, son muy baratos y existe la posibilidad de interconectar una gran cantidad de microprocesadores dentro de un sistema.

El multiprocesamiento mejora la seguridad del sistema; en cuanto ocurre una falla o error en una parte tiene un efecto limitado en el resto del sistema. Si una falla hace que un procesador deje de funcionar, se puede asignar un segundo procesador para ejecutar las funciones del procesador inhabilitado.

[1] M. Morris Mano, ***Arquitectura de computadoras***, 3ª ed., México, Editorial Prentice Hall Hispanoamericana, 1994, p. 563.

1.2 ¿QUÉ ES UN PROCESADOR?

El multiprocesador significa una unidad de procesamiento central (CPU) o una unidad de procesador de entrada salida IOP.

Los componentes del procesador incluyen la unidad de control, los registros, la unidad aritmética y lógica (ALU), la memoria del procesador y sus canales.

Los procedimientos de la computadora son ejecutados por la ALU bajo la dirección de la unidad de control. Estos procedimientos, juntos con los datos que han de ser procesados, residen en la memoria del procesador mientras que están siendo ejecutados. Los registros sirven para contener los datos que se están usando para obtener los valores intermedios, durante el procedimiento. Los canales se utilizan para transmitir los procedimientos y los datos hacia o desde la memoria del procesador, a medida que los requiere.

Los componentes del procesador pueden disponerse de maneras diferentes para suministrar una variedad de posibilidades y velocidades de procesamiento. Se denomina multiprocesador debido a que una memoria del procesador única se comparte por más de un conjunto de componentes de cada uno de otros procesadores. Así, una computadora es capaz de procesar múltiples instrucciones simultáneamente; por lo contrario, los monoprocesadores tratan sencillamente las instrucciones de una a una.

1.3 CLASIFICACIÓN DE LOS MULTIPROCESADORES

Los multiprocesadores se clasifican como:

- SISD (*Single Instruction, Single Data*). Flujo de sólo una instrucción.
- MISD (*Multiple Instruction, Single Data*). Flujo de múltiples instrucciones.
- SIMD (*Single Instruction, Multiple Data*) Flujo de una sola instrucción.
- MIMD (*Multiple Instruction stream, Multiple Data stream*). Flujo de múltiples instrucciones, flujo de múltiples datos.

SISD (*Single Instruction, Single Data*). Flujo de sólo una instrucción, flujo de un solo dato. Es el modelo que siguen la mayoría de las computadoras. Son equipos con un único procesador en los que se procesan algoritmos secuenciales sobre un conjunto de datos (se muestra en la figura 1-1). Se ejecutan los pasos uno a uno, no haciendo uso del *paralelismo*.

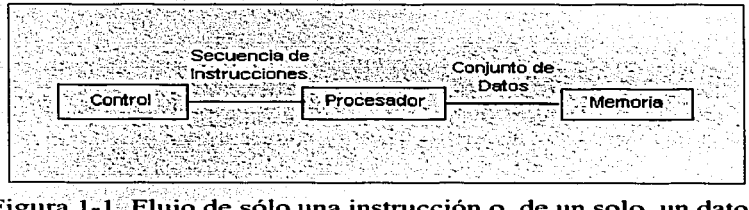


Figura 1-1 Flujo de sólo una instrucción o de un solo un dato [2]

MISD (*Multiple Instruction, Single Data*). Flujo de múltiples instrucciones. Esta es una organización que algunas veces utiliza múltiples instrucciones que operan sobre un único flujo de datos en paralelo. En este modelo las

[2] Véase www.aguila.itamx/bdd/unit.7htm.

computadoras disponen de N procesadores, cada uno con su unidad de control que comparten una unidad de memoria. Estos procesadores operan simultáneamente. Aunque todos los procesadores operen sobre los mismos datos, pueden realizar operaciones diferentes (se muestra en la figura 1-2).

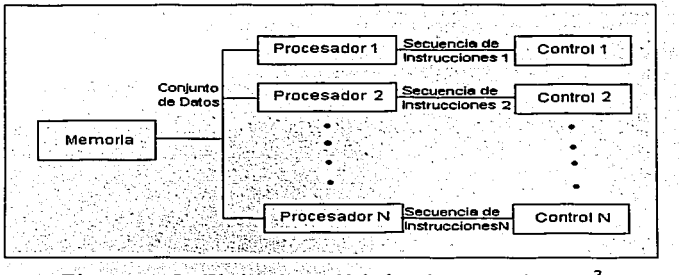


Figura 1-2 Flujo de múltiples instrucciones [3]

SIMD (*Single Instruction, Multiple Data*). Flujo de una sola instrucción o flujo de múltiples datos. Estos son típicamente los procesadores vectoriales y las computadoras en arreglo, en los cuales una sola instrucción puede operar sobre diferentes datos en diferentes unidades de ejecución (se muestra en la figura 1-3). Un equipo basado en un modelo SIMD tiene N procesadores idénticos, cada uno con una memoria local propia, trabajando con una única secuencia de instrucciones.

[3] Véase www.aguila.itamx/bdd/unit.7htm.

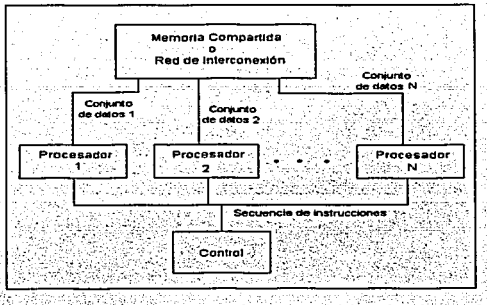


Figura 1-3 Flujo de una sola una instrucción ó de múltiples datos [4]

Todos los procesadores ejecutan la misma instrucción, aunque sobre distintos datos. En ocasiones puede ser necesario que sólo un subconjunto de procesadores ejecuten determinada instrucción, y será esta misma instrucción la que lleve codificada si el procesador debe permanecer *activo* o *inactivo*. Los procesadores que queden *inactivos* deberán esperar a la siguiente instrucción.

Para la ejecución de las tareas es necesario que los procesadores puedan comunicarse, usando memoria compartida o una red de interconexión.

MIMD (Multiple Instruction, Multiple Data). Flujo de múltiples instrucciones. Es la ejecución simultánea de múltiples y distintas instrucciones que operan sobre varios flujos de datos. Esta clase incluye los multiprocesadores de diferentes tipos. Es el más general y más potente de las computadoras en paralelo (se muestra en la figura 1-4). Se tienen N programas distintos, N grupos de datos y N procesadores con una unidad de control, memoria local y sus unidades aritméticas y propias lógicas.

[4] Véase www.aguila.itamx/bdd/unit.7htm.

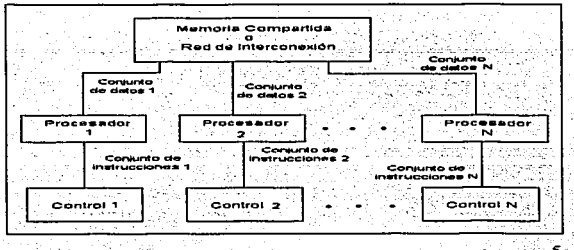


Figura 1-4 Flujo de múltiples instrucciones [5]

Cada procesador trabajará con las instrucciones que le envía la unidad de control, por lo que los procesadores pueden utilizar programas diferentes sobre datos también diferentes. Los procesadores trabajan de forma asíncrona, y se comunican entre sí bien, usando memoria compartida (*multiprocesadores* o *máquinas de acoplamiento fuerte*) o una red de interconexión (*multiordenadores* o *máquinas de acoplamiento débil*).

Si los procesadores se encuentran próximos entre sí, se habla de un multiordenador; pero si se encuentran distantes se habla de un sistema distribuido y entonces se tendrá que tener en consideración el número de intercambios entre procesadores.

Las computadoras MIMD se usan para resolver problemas con estructura irregular, más difíciles de diseñar, evaluar e implementar.

Un algoritmo asíncrono hace que una serie de procesos trabajen simultáneamente en un algoritmo paralelo; un procesador comienza los

[5] Véase www.aguila.itamx/bdd/unit7.htm.

procesos que generan tareas que se ejecutan en otros procesadores. Estos procesos o tareas son asignados a procesadores desocupados. Si no se encuentran procesadores libres se incluye la tarea en una cola de espera. Si por el contrario no hay suficientes tareas, se crea una cola de procesadores.

ORGANIZACIÓN DE LA MEMORIA DE MULTIPROCESADORES

Los multiprocesadores se clasifican, por la manera que se organiza su memoria y su organización, de la siguiente forma:

- Multiprocesador de memoria compartida
- Multiprocesador de memoria distribuida

Multiprocesador de memoria compartida. Un sistema multiprocesador con memoria compartida común se clasifica como multiprocesador de memoria compartida, esto no evita que cada procesador tenga su memoria local y la mayoría de los procesadores con memoria compartida proporcionan una memoria caché con cada CPU, hay una memoria global que pueden acceder todas CPU y por lo tanto, puede compartirse entre las CPU al colocarla entre la memoria global común.

Multiprocesador de memoria distribuida. Cada elemento del microprocesador en un sistema de memoria distribuida tiene su propia memoria local privada, los procesadores se enlazan mediante un esquema de conmutación diseñado para dirigir información de un procesador a otro, a través de un esquema de paso de mensajes. Los procesadores proporcionan

programas y datos a otros procesadores en paquete, un paquete consta de una dirección, el contenido de datos y algún código de detección de error. Los paquetes que proporcionan a un procesador específico, toman el primer procesador disponible del sistema de comunicación usado. Los sistemas de memoria distribuida son más eficientes cuando la interacción entre las tareas es mínima, mientras que los sistemas acoplados con precisión pueden tolerar un mayor grado de interacción entre tareas.

SIMD con Memoria Compartida. Este tipo de máquinas es también conocido como SM SIMD (*Shared Memory Single Instruction Multiple Data*), o como PRAM (*Parallel Random Access Machine*).

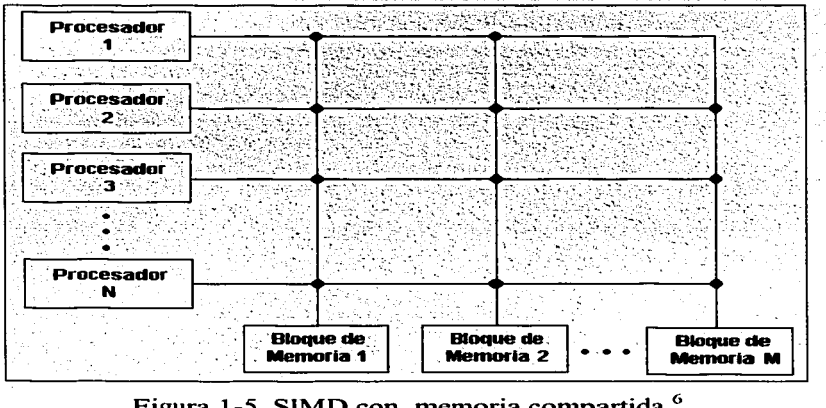


Figura 1-5 SIMD con memoria compartida [6]

Se utiliza una memoria común para todos los procesadores, logrando que mediante ésta puedan comunicarse entre sí (se muestra en la figura 1-5).

[6] Véase www.aguila.itamx/bdd/unit7.htm.

Las computadoras SIMD se pueden clasificar según sus procesadores que puedan acceder simultáneamente a una misma posición de memoria; se presentan en cuatro tipos:

1. **EREW**, Exclusive-Read Exclusive-Write. Dos procesadores no podrán escribir o leer en una misma posición de memoria a la vez.

2. **CREW**, Concurrent-Read Exclusive-Write. Varios procesadores podrán leer en una misma posición de memoria al mismo tiempo, pero no escribir.

3. **ERCW**, Exclusive-Read Concurrent-Write. Varios procesadores podrán escribir en una misma posición de memoria simultáneamente, pero no podrá leer una misma dirección a la vez.

4. **CRCW**, Concurrent-Read Concurrent-Write. Se podrá escribir o leer en una misma posición de memoria simultáneamente.

En cualquier caso, la lectura simultánea no se debe suponer ningún problema, pero no se puede decir lo mismo de la escritura. Para resolver los problemas de escritura se pueden utilizar varios criterios:

1. Dejar escribir sólo al que tenga el número de identificación menor.

2. Sólo dejar escribir si todos los procesadores van a escribir lo mismo, en caso contrario denegar la escritura a todos.

3. Almacenar la suma de todas las escrituras.

El peor de los modelos para memoria compartida se toma en cuenta a la hora de diseñar un algoritmo por lo que se debe evitar conflictos de acceso a memoria. Entonces se considera el número de accesos múltiples que se van a realizar sobre una misma dirección de memoria, o bien con todos los procesadores, o sólo un grupo de éstos.

N accesos múltiples, siendo N el número de procesadores. Si todos acceden a la misma dirección de memoria se utiliza un proceso de difusión:

1. P_1 (el procesador 1) lee la posición de memoria y comunica su valor a P_2.

2. P_1 y P_2 comunican simultáneamente el valor leído a P_3 y P_4.

3. Estos cuatro procesadores comunican el valor leído de la posición de memoria a otros tantos procesadores.

Este proceso se va repitiendo hasta que los N procesadores conocen el contenido de la memoria.

Para realizar una escritura se utiliza un proceso equivalente que funciona sólo si todos los procesadores pretenden escribir el mismo valor.

1. Para $1 \leq i \leq N/2$, si los valores a escribir son iguales en los procesadores P_i y $P_{i+N/2}$, entonces P_i sustituye la variable intermedia *i* por *verdadero*.

2. Para 1 ≤i ≤N/4, si la variable intermedia *i* e *i+N/4* son verdaderas y los valores de P_i y $P_{i+N/4}$ son iguales, P_i substituye su variable intermedia *i* por *verdadero*.

Este proceso se repite *log N* veces, conociendo entonces si se puede escribir o no en la memoria.

M accesos múltiples, siendo N el número de procesadores y M<N. Normalmente no va ha ser necesario que todos los procesadores accedan a una misma dirección de memoria, si se comparte M posiciones de memoria se asocia a éstas 2N-2 palabras que forman un árbol con N bifurcaciones, cada una asociada a un procesador, y que tiene por raíz de éste a la propia M.

Tanto para la lectura como para la escritura, las solicitudes de los procesadores avanzan hacia la raíz de la misma forma que se hace para los N accesos múltiples (caso anterior).

Cada vez que un procesador accede a una posición de memoria necesita establecer un camino físico hasta ésta, lo que supone un aumento de coste en circuitería se expresa como f(M).

1.4 CARACTERÍSTICAS DE LOS MULTIPROCESADORES

Existen varias semejanzas entre sistemas multiprocesadores y multicomputadoras, porque ambos soportan operaciones concurrentes; pero existe una diferencia importante entre un sistema de computadoras múltiples y un sistema con procesadores múltiples. Las computadoras se interconectan

unas con otras mediante líneas de comunicación para formar una red de computadoras; dicha red consiste en varias computadoras autónomas que pueden comunicarse o no con otras.

Un multiprocesador mejora la confiabilidad del sistema, por lo que una falla o error en una parte tiene un efecto limitado en el resto del sistema. Si una falla hace que un procesador deje de funcionar, puede asignarse un segundo procesador para ejecutar las funciones del procesador para ejecutar las funciones del procesador inhabilitado.

1.5 VENTAJAS DE LOS MULTIPROCESADORES

Rendimiento y potencia de cálculo. Se encarga de la ejecución de múltiples procesadores de una aplicación con la posibilidad de obtener ganancias significativas de velocidad. Los problemas que suponen las interacciones más extensas o más frecuentes entre procesadores pueden resolverse más rápidamente que un sistema distribuido ya que el ancho de banda de comunicación entre procesadores es más elevado.

Tolerancia a fallos. La redundancia de los multiprocesadores puede emplearse para aumentar la disponibilidad, eliminar puntos de fallo y una alta seguridad ya que un multiprocesador, a diferencia de un monoprocesador, puede trabajar en forma continua, aunque falle una.

Flexibilidad. Un sistema multiprocesador podría tener la capacidad para reconfigurarse dinámicamente y ajustarse de tal modo que optimice diferentes objetivos para diferentes aplicaciones, tales como aumento de productividad,

ganancias de velocidad en las aplicaciones o incluso ciertos aspectos de tolerancia a fallos.

Crecimiento modular. Hasta cierto punto, un diseño modular de un sistema puede adaptarse a las necesidades de una instalación especifica añadiéndole exactamente el tipo de componente que más probablemente alivie un "cuello de botella", tal como procesadores, memorias o dispositivos de E/S.

Especialización funcional. Se pueden añadir procesadores funcionalmente especializados para mejorar el rendimiento de aplicaciones particulares.

Costo / rendimiento. Los microprocesadores comerciales con relaciones costo/rendimiento y con varias órdenes de magnitud inferiores a la de los supercomputadores pueden estructurarse en multiprocesadores con una relación costo/efectividad adecuada a un amplio rango de aplicaciones.

CAPITULO DOS

ESTRUCTURAS DE INTERCONEXIÓN EN LOS MULTIPROCESADORES

La característica principal de un sistema multiprocesador es la posibilidad de que cada procesador comparta un conjunto de módulos de memoria principal y dispositivos de E/S. Los componentes que forman un sistema multiprocesador son las CPU, los IOP [7] conectados a dispositivos de entrada-salida y una unidad de memoria que puede dividirse en varios módulos separados. La interconexión entre los componentes puede tener diferentes configuraciones físicas, dependiendo de la cantidad de trayectorias de transferencia disponibles entre los procesadores y la memoria, en un sistema de memoria compartida o entre elementos de procesamiento, en un sistema de memoria distribuida. Existen varias formas físicas disponibles para establecer una red de interconexión.

En general, la ruta de interconexión del multiprocesador tiene una influencia dominante sobre el ancho de banda y el punto de saturación para las comunicaciones de los sistemas. Algunas de las arquitecturas de los multiprocesadores son las siguientes:

1. Canal común de tiempo compartido
2. Memoria de multipuertos
3. Conmutador de barra de cruz
4. Red de conmutación de etapas múltiples
5. Sistema de hipercubo

[7] Procesador entrada-salida.

2.1 CANAL COMÚN DE TIEMPO COMPARTIDO

Es el sistema de interconexión más simple para múltiples procesadores en un camino de comunicación común que conecta a todas las unidades funcionales. Un sistema multiprocesador de canal común consta de varios procesadores conectados mediante una trayectoria común a una unidad de memorias. Un canal de tiempo compartido para cinco procesadores se muestra en la figura 2-1. En cualquier momento, un procesador puede comunicarse con la memoria o con otro procesador. Las operaciones de transferencia las controla el procesador que controla el canal en ese momento. Cualquier otro procesador que desea iniciar una transferencia debe determinar primero el estado de disponibilidad de canal, y sólo después de que el canal queda disponible, puede el procesador direccionar la unidad destino para iniciar la transferencia. Se presenta un comando para informar a la unidad destino cuál operación se va a ejecutar. La unidad que recibe reconoce su dirección en el canal y responde a las señales de control de la unidad que las envía, después de lo cual se inicia la transferencia.

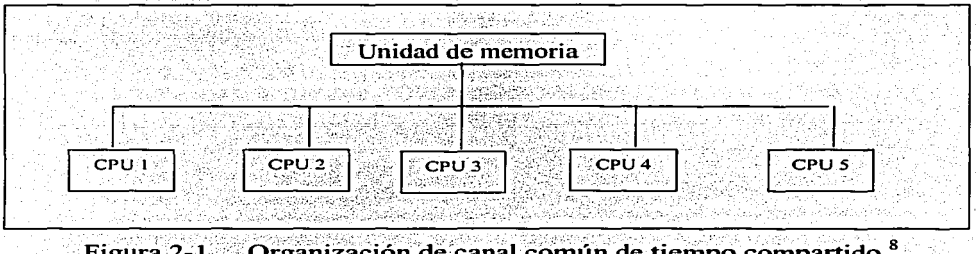


Figura 2-1 Organización de canal común de tiempo compartido [8]

[8] Véase *supra* nota 1, p.528.

El sistema puede mostrar conflictos de transferencia, dado que todos los procesadores comparten un canal común y este conflicto se resuelve al incorporar un controlador de canal que establezca prioridades entre las unidades solicitantes. Un sistema de canal común único está limitado a una transferencia a la vez. Esto significa que cuando otro procesador se está comunicando con la memoria, todos los otros procesadores están ocupados con operaciones internas o deben de estar inactivos en espera de canal; en consecuencia, la velocidad de transferencia total dentro del sistema está limitada a la velocidad de la trayectoria única. Con frecuencia, los procesadores en el sistema pueden mantenerse ocupados por medio de la implantación de dos o más canales independientes, para permitir transferencias de canal simultáneas y múltiples. Sin embargo, esto incrementa el costo y la complejidad del sistema.

En la figura 2-2 se muestran varios canales locales, cada uno conectado a su propia memoria local y a uno o más procesadores. Cada canal local puede estar conectado a una CPU, un IOP o una combinación de procesadores. Un controlador de canal de sistema enlaza cada canal local a un canal de sistema común. Los dispositivos de E/S conectados al IOP local, al igual que a la memoria local, están disponibles para el procesador local. La memoria conectada al canal de sistema común la comparten todos los procesadores. Si un IOP está conectado en forma directa del canal de sistema, los dispositivos de E/S conectados a él pueden quedar disponibles para todos los procesadores. Sólo un procesador puede comunicarse con la memoria compartida y otros recursos comunes a través del canal del sistema, en cualquier momento dado. Los otros procesadores se mantienen ocupados comunicándose con su memoria local y sus dispositivos de E/S.

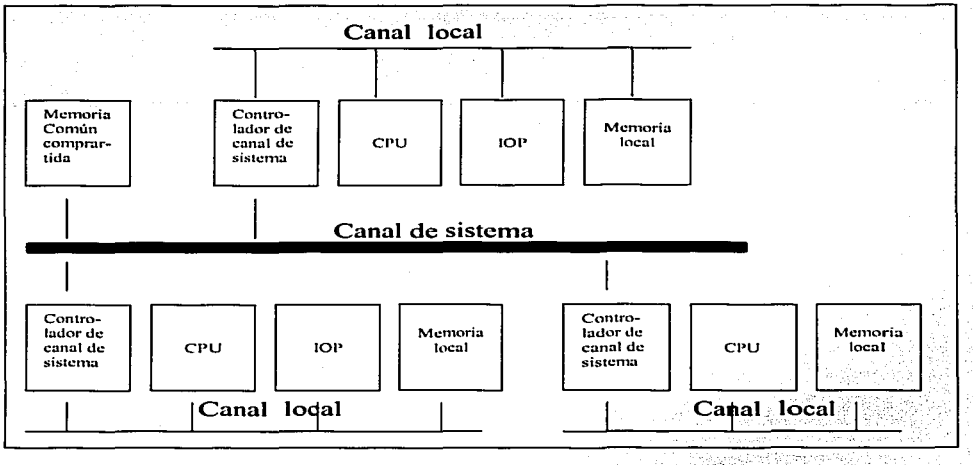


Figura 2-2 Estructura de canal de sistema para multiprocesadores [9]

2.2 MEMORIA MULTIPUERTO

El sistema de memoria multipuerto utiliza canales separados entre cada módulo de memoria y cada CPU (se muestra en la figura 2-3 para cuatro CPU y cuatro módulos de memoria). Cada canal de procesador se encuentra conectado a cada módulo de memoria. Un canal de procesador consta de la dirección, los datos y las líneas de control requeridas para comunicarse con la memoria. El módulo de memoria tiene cuatro puertos y cada puerto aloja uno de los canales.

[9] Véase *supra* nota 1, p.529.

Por lo tanto, el módulo tiene lógica de control interna para determinar cuál puerto tendrá acceso a memoria en cierto momento.

Los conflictos de acceso a memoria se resuelven al asignar prioridades fijas a cada puerto de la memoria. La prioridad para acceso a memoria asociada con cada procesador puede establecer la posición física del puerto que ocupa su canal en cada módulo. Por ello tanto, la CPU 1 tendrá prioridad sobre la CPU 2; la CPU 2 tendrá prioridad sobre la CPU 3, y la CPU 4 tendrá la prioridad menor. Esta estructura de interconexión, por lo general, es apropiada para sistemas con un número reducido de procesadores.

La ventaja de la organización de memoria multipuertos es la alta velocidad de transferencia que puede conseguirse debido a las trayectorias múltiples entre los procesadores y la memoria.

La desventaja es que requiere una lógica de memoria cara y una gran cantidad de cables y conectores.

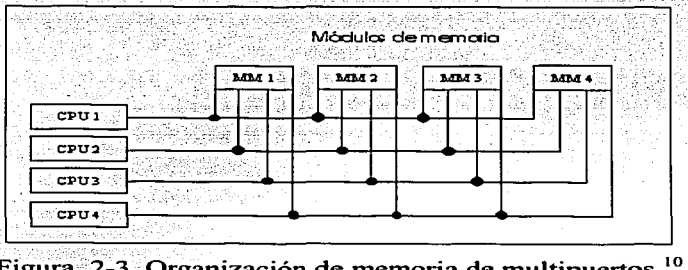


Figura 2-3 Organización de memoria de multipuertos [10]

[10] Véase *supra* nota 1, p.530.

2.3 CONMUTADOR DE BARRA DE CRUZ

La organización de conmutador de barra de cruz consta de varios puntos de cruz que se colocan en interacciones entre los canales del procesador y las trayectorias del módulo de memoria. La figura 2-4 muestra una interconexión de conmutador de barra de cruz entre cuatro CPU y cuatro módulos de memoria. El cuadro pequeño en cada cruz es un conmutador que determina la trayectoria de un procesador a un módulo de memoria.

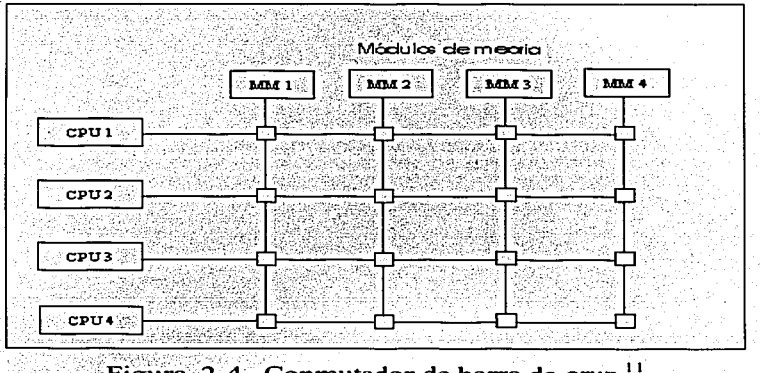


Figura 2-4 Conmutador de barra de cruz [11]

Cada punto del conmutador tiene lógica de control para inicializar la trayectoria de transferencia entre un procesador y la memoria. Examina la dirección que está colocada en el canal para determinar si está direccionando su módulo particular, y resuelve solicitudes múltiples de acceso al mismo módulo de memoria en base a una prioridad predeterminada.

[11] Véase *supra* nota 1, p.531.

El diseño funcional de un conmutador de barra de cruz conectado a un módulo de memoria se muestra en la figura 2-5. Este circuito consiste en multiplexores que seleccionan los datos, la dirección y el control de una CPU para la comunicación con el módulo de memoria, los niveles de prioridad se establecen mediante la lógica de arbitraje, para seleccionar una CPU cuando dos o más de ellas intentan accesar a la misma memoria, los multiplexores se controlan con el código binario que genera un codificador de prioridad dentro de la lógica de arbitraje.

Una organización de conmutador de barra de cruz soporta las transferencias simultáneas de todos los módulos de una memoria porque hay una trayectoria separada asociada con cada módulo, una desventaja es que la circuitería necesaria para poder implementar el conmutador resulta demasiado grande y complejo.

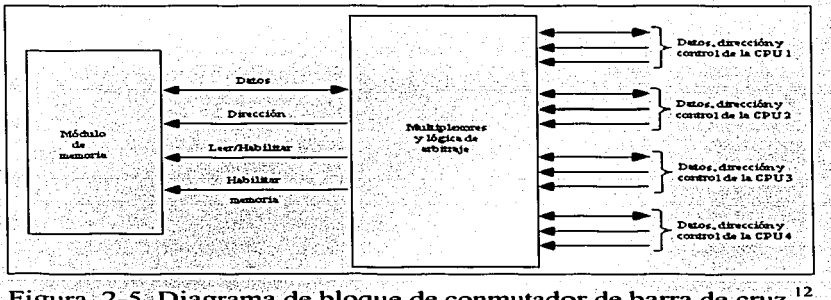


Figura 2-5 Diagrama de bloque de conmutador de barra de cruz [12]

[12] Véase *supra* nota 1, p.532.

2.4 RED DE CONMUTACIÓN DE ETAPAS MÚLTIPLES

El componente básico de una red de etapas múltiples es un conmutador de intercambio de dos entradas, dos salidas como se muestra en la figura 2-6, el conmutador 2 x 2 tiene dos entradas, A y B y dos salidas, 0 y 1. El conmutador tiene la capacidad de conectar la entrada A a cualquiera de las salidas, la terminal B del conmutador se comporta de manera similar, el conmutador también tiene la capacidad para proporcionar arbitraje entre solicitudes en conflicto. Si las entradas A y B solicitan la misma terminal de salida, sólo se conectará una de ellas mientras la otra se bloquea.

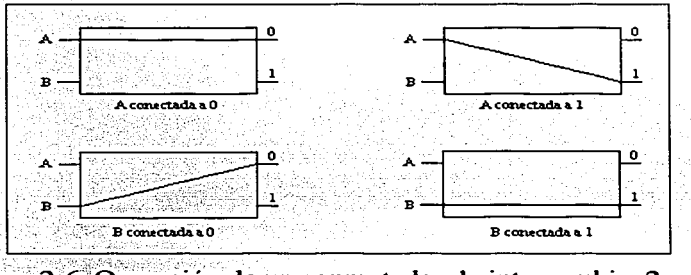


Figura 2-6 Operación de un conmutador de intercambio 2 x 2 [13]

Al utilizar el conmutador 2 x 2 como un bloque de construcción, se puede desarrollar una red de etapas múltiples que controle la comunicación entre varias fuentes y destinos. En la figura 2-7 se muestra el árbol binario donde los dos procesadores P_1 y P_2 se conectan mediante conmutadores a ocho

[13] Véase *supra* nota 1, p.533.

módulos de memoria marcados en binario del 000 a 111. La trayectoria de una fuente destino la determinan los bits binarios del número destino.

El primer bit del número destino determina la salida del conmutador en el primer nivel, el segundo bit especifica la salida del conmutador en el tercer nivel (un ejemplo: para conectar P_1 a la memoria 101, es necesario formar una trayectoria de P_1 a la salida 1 en el conmutador del primer nivel, a la salida 0 en el conmutador de segundo nivel y a la salida 1 en el conmutador de tercer nivel y tomando en cuenta P_1 y P_2 pueden conectarse a cualquiera de las ocho memorias). Sin embargo, ciertos patrones de solicitud no pueden satisfacerse en forma simultánea. Por lo tanto, si P_1 está a uno de los destinos 000 a 011, P_2 puede conectarse a sólo uno de los destinos 100 al 111.

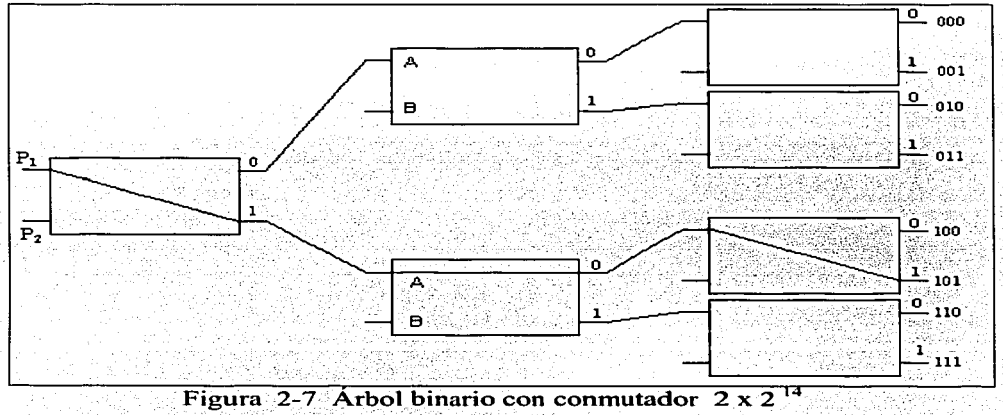


Figura 2-7 Árbol binario con conmutador 2 x 2 [14]

[14] Véase *supra* nota 1, p.533.

Se han propuesto muchas topologías diferentes para las redes de conmutación de etapas múltiples, con la finalidad de controlar la comunicación procesador-memoria en un sistema de multiprocesador de memoria compartida o para controlar la comunicación entre los elementos de procesamiento en un sistema de memoria distribuida. Una de estas topologías es la red de conmutación omega que se muestra en la figura 2-8, esta configuración existe una trayectoria de cada fuente a cualquier destino particular. No obstante, algunos patrones de solicitud no pueden conectarse en forma simultánea.

En un sistema multiprocesador de memoria compartida, la fuente es un procesador y el destino es un módulo de memoria. La primera pasada por la red establece la trayectoria. Se utilizan pasadas sucesivas para transferir la dirección dentro de la memoria y después de transferir los datos en cualquier dirección, dependiendo de si la solicitud es una lectura o una escritura. En un sistema de multiprocesador de memoria distribuida la fuente y el destino son elementos de procesamiento. Después de que se establece la trayectoria, el procesador fuente transfiere un mensaje al procesador destino.

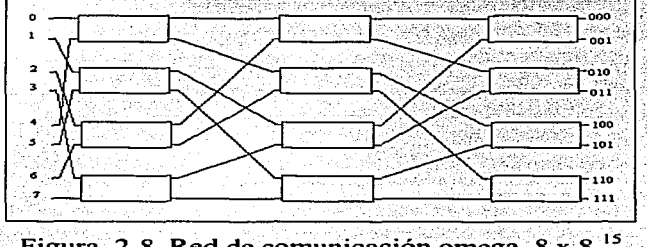


Figura 2-8 Red de comunicación omega 8 x 8 [15]

[15] Véase *supra* nota 1, p.534.

2.5 SISTEMA DE HIPERCUBO

La estructura de procesador de hipercubo o cubo n binario es un sistema de memoria distribuida compuesto de $N=2^n$ procesadores interconectados en un cubo binario de n dimensiones. Cada procesador forma un nodo del cubo. Aunque se acostumbra decir que cada nodo tiene un procesador, de hecho no sólo contiene una CPU, sino también memoria local e interfaces de E/S. Cada procesador tiene trayectorias de comunicación directa a n procesadores vecinos. Estas trayectorias corresponden a los lados del cubo. Existen 2^n direcciones binarias de n bits distintas que pueden asignarse a los procesadores y cada dirección de procesador es diferente de cada una de sus n vecinas siendo exactamente una posición de bit.

En la figura 2-9 se muestra la estructura de hipercubo para n= 1, 2, 3; una estructura de un cubo tiene n=1 y 2^n= 2. contiene dos procesadores interconectados mediante una trayectoria única. Una trayectoria de dos hipercubos tiene 2 y 2^n = 4, contiene cuatro nodos interconectados en un cuadro. Una estructura de tres cubos tiene ocho nodos interconectados como un cubo.

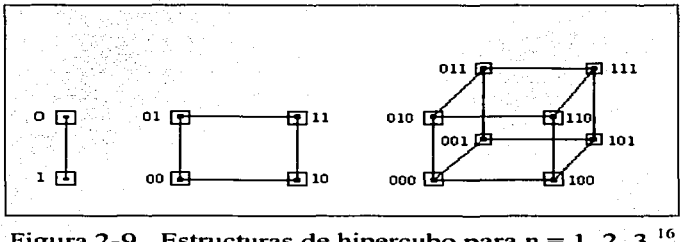


Figura 2-9 Estructuras de hipercubo para n = 1, 2, 3 [16]

[16] Milan Milenković, ***Sistemas operativos***, 2ª ed., España, Editorial Mc Graw-Hill, 1994, p.581.

Una estructura de n cubos tiene 2^n nodos y un procesador residiendo en cada nodo. A cada nodo se le asigna una dirección binaria, de manera que las direcciones de dos vecinos sean diferentes en exactamente una posición de bit. Por ejemplo, los tres vecinos del nodo con la dirección 100 en una estructura de tres cubos son 000, 110, y 101. Cada uno de estos números binarios es diferente de la dirección 100 en un valor de bit.

Enrutar mensajes a través de una estructura de n cubos puede requerir de uno a n enlaces de un nodo fuente a un nodo destino. Por ejemplo, en una estructura de tres cubos, el nodo 000 puede comunicarse directamente con el nodo 001; debe cruzar al menos dos enlaces para comunicarse con 011(de 000 a 001 a 011 o de 010 a 011). Es necesario recorrer al menos tres enlaces para comunicarse del nodo 000 al nodo 111. Puede desarrollarse un procedimiento de direccionamiento al calcular la OR exclusiva de la dirección del nodo fuente con la dirección del nodo destino.

El valor binario resultante tendrá un bit que corresponda a los ejes en los cuales son diferentes los dos nodos. Después el mensaje se envía por algunos de los ejes. Por ejemplo, en una estructura de tres cubos, un mensaje en 010 que va a 001 produce una OR exclusiva de las dos direcciones iguales a 011. El mensaje puede enviarse por el segundo eje a 000 después de recorrer el tercer eje a 001

Un representante de la arquitectura de hipercubo es el complejo de computadoras de Intel iPSC. Consta de 128(n=7) microcomputadoras conectadas mediante canales de comunicación. Cada nodo consta de una CPU, un procesador de punto flotante, memoria local, y unidades de interface de comunicación serial. Los nodos individuales operan en forma independiente

sobre los datos almacenados en la memoria local, de acuerdo con los programas residentes. Los datos y los programas a cada nodo provienen de un sistema de paso de mensajes, de otros nodos o del administrador del cubo.

Se desarrollan y se compilan los programas de aplicaciones en el administrador del cubo y después se cargan en los nodos individuales. Los cálculos se distribuyen por el sistema y se ejecutan en forma concurrente.

CAPITULO TRES

SISTEMAS OPERATIVOS MULTIPROCESADORES

Las primeras investigaciones y las implementaciones de los primeros multiprocesadores estuvieron dedicados a la demostración de conceptos. Los principales objetivos de investigación eran las arquitecturas y los modos de explotar el paralelismo para aumentar la velocidad (para aplicaciones especializadas en cálculo). Debido a la relativa falta de experiencia y al ilimitado uso de los multiprocesadores, queda mucho trabajo por hacer en cuanto al descubrimiento de los principios y sobre la metodología de diseño para sistemas operativos en multiprocesadores.

Existe una complejidad en el sistema operativo cuando múltiples procesadores trabajan simultáneamente. La complejidad se origina por la necesidad de que el sistema operativo tenga que soportar múltiples tareas asíncronas que se ejecutan concurrentemente.

Un sistema multiprocesador necesita técnicas para la utilización eficiente de recursos y debe proporcionar esquemas de entrada-salida y equilibrado de carga del procesador. Una razón principal para utilizar un sistema multiprocesador estriba en proporcionar una fiabilidad efectiva y una degradación ordenada (graceful degradation) cuando ocurre un fallo.

Existen tres organizaciones tipos de sistemas operativos multiprocesadores:

- Supervisores separados.
- Maestro/esclavos.
- Simétrico.

3.1 SUPERVISORES SEPARADOS

En sistemas supervisores separados, cada nodo contiene un sistema operativo aparte que gestiona el procesador local, la memoria y los recursos de E/S. La compartición de recursos ocurre a un nivel más alto, por ejemplo, a través de una estructura compartida de archivos.

Cada procesador sirve sus propias necesidades. Sin embargo, existen interacciones entre procesadores, es necesario que parte del código supervisor sea duplicado para proporcionar copias separadas a cada procesador. Aunque cada supervisor tiene su propio conjunto de tablas privadas, algunas tablas son comunes y compartidas por el sistema completo. Esto crea problemas de acceso a tablas. El método utilizado para acceder a los recursos compartidos depende del grado de acoplamiento entre los procesadores. El sistema operativo con supervisor separado no es tan sensible a fallos catastróficos como el maestro/esclavo. También cada procesador tiene su propio conjunto de dispositivos de entrada-salida y archivos, y cualquier reconfiguración de E/S requiere habitualmente intervención manual y posiblemente conmutación manual.

Un ejemplo de supervisores separados lo proporcionan los sistemas hipercubos, que debido a su estructura regular y repetitiva construida a partir de bloques idénticos, tienden a replicar copias casi idénticas de un núcleo en cada nodo. El núcleo proporciona servicios tales como gestión de procesos y de memorias locales e implementa primitivas de paso de mensaje.
Las funciones de almacenar y reexpedir y de entregar mensajes pueden implementarse dentro del núcleo como una tarea del sistema. El paralelismo

dentro de las aplicaciones se logra rutinariamente subdividiéndolas en subtareas que se ejecutan sobre nodos diferentes.

3.2 MAESTRO/ESCLAVOS

En el método maestro/esclavos, un procesador se dedica a ejecutar el sistema operativo. Los restantes procesadores son generalmente idénticos y forman un depósito de procesadores computacionales.

El procesador maestro planifica el trabajo y controla la actividad de los esclavos. La mayoría de las estructuras de datos del sistema operativo están controladas por el procesador maestro y suelen estar almacenadas en su memoria privada.

Los procesadores esclavos pueden ser capaces de procesar directamente algunas consultas locales simples, pero la mayoría de los servicios del sistema operativo son proporcionados por el maestro.

Tiene la capacidad de permitir el paralelismo dentro de una aplicación mediante la asignación a ella de múltiples esclavos. Sin embargo, poco o ningún paralelismo es posible para el sistema operativo, ya que su ejecución está limitada a un solo nodo.

Los sistemas maestro/esclavos son relativamente sencillos de desarrollar. Añadiéndole la planificación de esclavos, un sistema operativo monoprocesador serie puede ser adaptado con bastante facilidad para operación multiprocesador maestro/esclavos. El sistema resultante tiene una

escalabilidad limitada, ya que la existencia de un único nodo maestro suele convertirse en un cuello de botella.

3.3 SIMÉTRICO

El multiprocesamiento simétrico es la organización donde todos los procesadores son funcionalmente idénticos, es más complicada de implementar y la más poderosa. Todos los procesadores son idénticos. El sistema operativo administra un grupo de procesadores idénticos, cualquiera de los cuales puede ser utilizado para controlar cualquier dispositivo de entrada/salida , o hacer referencia a cualquier unidad de almacenamiento.

Como varios procesadores pueden estar ejecutando al mismo tiempo el sistema operativo, son necesarios el código reentrante y la exclusión mutua. A causa de la simetría, es posible equilibrar la carga de trabajo de forma más precisa que en los otros tipos.

El hardware y software de resolución son importantes. Los conflictos entre los procesadores que intentan acceder al mismo tiempo, suelen ser resueltos por el hardware. Los conflictos en el acceso a las tablas del sistema los suele resolver el software.

Los sistemas de multiprocesamiento simétricos son en general, los más confiables, cuando falla un procesador, el sistema operativo lo retira de su grupo de procesadores disponibles y notifica la dificultad a su operador. El sistema puede seguir trabajando a niveles reducidos, mientras se repara el procesador que falló.

Un proceso en ejecución en un sistema de multiprocesadores simétricos puede ser ejecutado en diferentes momentos por cualquiera de los procesadores equivalentes. Todos los procesadores pueden cooperar en la ejecución de un proceso determinado.

En la forma más sencilla de organización simétrica, conocida como *maestro flotante*, sobre los requisitos de carga de la carga de trabajo y la disponibilidad de los procesadores, el sistema operativo se ejecuta en diferentes procesadores en instantes diferentes.

Temporalmente, el procesador que ejecuta el sistema operativo juega un papel especial y actúa como maestro en el sentido de que planifica el trabajo de los demás. El sistema operativo no esta ligado a ningún procesador específico; flota de un procesador a otro por eso se le conoce como maestro flotante. Puesto que el propio sistema operativo es en gran medida monolítico, muy poco de su código, si alguno, se ejecuta en paralelo.

3.4 CONFIGURACIONES DE SISTEMAS OPERATIVOS PARA MULTIPROCESADORES

Sistema operativo de supervisores separados:

1. El "maestro" separa de un procesador a otro, aunque varios procesadores pueden estar ejecutando rutinas de servicio del supervisor al mismo tiempo.

2. Este tipo de sistema puede lograr mejor balance de carga sobre todo tipo de recursos.

3. Los conflictos en las peticiones de servicios se resuelven por prioridades que pueden ser estáticas o dinámicas.

4. La mayor parte del código supervisor debe ser reentrante, ya que varios procesadores pueden ejecutar la misma rutina de servicio al mismo tiempo.

5. Pueden ocurrir conflictos de acceso a las tablas y retardos de bloqueo, pero no hay forma de evitar esto con supervisores múltiples; el punto importante es que se deben controlar de tal manera que la integridad del sistema quede protegida.

Sistemas operativos maestro/esclavos:

1. Tener un único procesador ejecutando el supervisor simplifica los problemas de conflicto y bloqueo en el control de tablas. El sistema completo es comparativamente inflexible. Este tipo de sistema necesita comparativamente software y hardware simples.

2. El sistema completo está sujeto a fallos catastróficos que requieren la intervención del operador para realizarlo cuando el procesador diseñado como maestro tiene un fallo o un error irrecuperable.

3. Este tipo de sistema operativo es más efectivo para aplicaciones especiales donde la carga de trabajo está bien definida o para sistemas asimétricos en

los que los subordinados tienen menos capacidad que el procesador maestro.

Sistema operativo simétrico:

1. Cada procesador se sirve sus propias necesidades. En efecto, cada procesador (supervisor) tiene su propio conjunto de equipos de E/S, archivos, etc.

2. Es necesario que parte del código supervisor sea reentrante o duplicado para proporcionar copias separadas a cada procesador.

3. El sistema operativo simétrico es tan sensible como maestro/esclavo; sin embargo, la reinicialización de un procesador individual que ha fallado será probablemente bastante difícil.

3.5 REQUISITOS DE SOFTWARE PARA MULTIPROCESADORES

Existen dos diferencias en el software escrito para ser ejecutado sobre un procesador múltiple del escrito para ser ejecutado sobre el entorno más familiar de un monoprocesador multiprogramado.

Un monoprocesador multiprogramado puede simular el entorno del procesador múltiple mediante la creación de múltiples "procesadores virtuales". Por ejemplo, un usuario UNIX pide rutinariamente la ejecución concurrente de programas múltiples con la salida de un programa "enchufada" como entrada a

otro. En este caso, cada programa se puede ver como si se ejecutara sobre un procesador virtual. A este nivel de la ejecución del programa hay pocas diferencias entre un sistema monoprocesador multiprogramado y un sistema multiprocesador. Sin embargo, la presencia de procesadores múltiples y otros componentes duplicados incrementa normalmente la cantidad de software administración que se debe proporcionar.

Un atributo arquitectónico que puede afectar a la programación en un sistema multiprocesador es la no homogeneidad. Si los procesadores centrales no son homogéneos, esto es, difieren funcionalmente, se deben tratar de forma diferente por el software. Por ejemplo si un procesador posee capacidad de emulación no poseída por otro, algunos programas solo se pueden ejecutar completamente sobre el procesador con la capacidad de emulación. Por lo tanto, los administradores de recursos software deben proporcionar mecanismos de distribución apropiados para tales programas. Otro ejemplo de complejidad software tiene lugar en un sistema con memoria principal asimétrica. En este caso, no todos los procesadores pueden acceder a la memoria. Esto complica el software del sistema operativo para la administración de recursos.

Hay una segunda fuente potencial de diferencias entre software del multiprocesador y del monoprocesador. Se encuentra en el propio estilo de programación para aplicaciones paralelas. La unidad básica de un programa en ejecución es un proceso, una unidad planificable independientemente (un programa secuencial) que se ejecuta sobre un procesador y usa recursos hardware y software. Un proceso se puede ejecutar concurrentemente con otros procesos, retardado (al menos lógicamente) sólo cuando necesite esperar para interactuar con uno o más procesos. Se puede decir que un programa paralelo consta de dos o más procesos que interactúan.

El potencial del multiprocesamiento se consigue aumentando su capacidad para el procesamiento paralelo. En un programa, el procesamiento paralelo se puede indicar explícita o implícitamente. Para indicar el paralelismo explícito de un programa, los usuarios deben disponer de los elementos de programación adecuados. El paralelismo implícito es detectado por el compilador. En este caso, el compilador analiza el programa fuente e identifica el flujo del programa.

En un sistema multiprocesador, la sincronización tiene una gran importancia ya que podría crear una elevada penalización. Si los mecanismos de sincronización no son eficientes y los algoritmos que los utilizan no están adecuadamente diseñados, el rendimiento del programa se podría degradar significativamente. En algunos procesadores, las primitivas de sincronización no están implementadas directamente en hardware o microcódigo. En un entorno donde los procesos necesitan una frecuente sincronización, esto se puede convertir en un importante cuello de botella.

Para ayudar al programador en el desarrollo de algoritmos paralelos eficientes se le proporcionan estructuras de control del programa. Se han identificado tres estructuras básicas no secuenciales de control de programas. Estas estructuras están caracterizadas por el hecho de que el programador sólo necesita centrarse sobre un programa pequeño y no sobre todo el control de computación.

El *primer ejemplo* es la organización basada en mensajes que se utilizó en el sistema operativo Cm*. En esta organización la computación se realiza mediante múltiples procesos homogéneos que se ejecutan independientemente e interactúan a través de mensajes.

El *segundo ejemplo* de una estructura de control es la estructura tarea (chore). En esta estructura todos los códigos se descomponen en unidades pequeñas. El proceso que ejecuta la unidad de código y el código mismo se denomina *tarea*. Una característica importante de una tarea, es que una vez que se comienza la ejecución no se detiene hasta que se completa. Para evitar esperas largas, las tareas son muy pequeñas. Tienen relativamente muy poca entrada y sólo hacen referencia a unos pocos objetos diferentes. No bloquean y no son interrumpibles.

Algunas tareas pueden incluir:

- La orden al disco para pedir la transferencia de una página de datos entre el disco y la memoria.
- El reconocimiento de la terminación de la transmisión de un sector de disco y la disponibilidad para la acción subsiguiente.

La *tercera estructura* de control no secuencial es la de los sistemas de producción, utilizados con frecuencia ahora en sistemas de inteligencia artificial. Las producciones son expresiones de la forma (antecedente, consecuente). Si la evaluación del antecedente booleano resulta verdad, se puede realizar el consecuente. En contraste con las tareas, los consecuentes con las producciones pueden o no incluir código capaz de provocar bloqueo.

En un sistema de producción se requieren cuatro estrategias de planificación:

a) Controlar la selección de antecedentes que se dejen evaluar a continuación.

b) Ordenar la ejecución de los antecedentes seleccionados.

c) Seleccionar el subconjunto de consecuentes ejecutables.

d) Ordenar la ejecución de los consecuentes seleccionados.

Las tres estructuras de control son todas compatibles con la ejecución paralela. El alto grado de concurrencia en un multiprocesador puede incrementar la complejidad del manejo de fallos, especialmente en el paso de recuperación. En un monoprocesador siempre es posible eliminar el paralelismo descapacitando las interrupciones y, si fuera necesario, parando la actividad de E/S. Para establecer una capacidad efectiva de recuperación de errores se necesita el software correspondiente. Este software, aun con la ayuda de mecanismo de hardware puede ser bastante complejo. Comprender el comportamiento de los procesos que se ejecutan en un sistema multiprocesador es más complejo que en un entorno monoprocesador. Aunque los programas paralelos no son demasiado complejos de implementar, existe un problema natural de no determinismo en los multiprocesadores.

3.6 FUNCIONES Y REQUISITOS DE LOS SISTEMAS OPERATIVOS MULTIPROCESADORES

Al igual que los sistemas operativos monoprocesadores, un sistema operativo multiprocesador gestiona los recursos disponibles y acrecienta la funcionalidad hardware para formar una abstracción que facilite la ejecución de programas y la interacción con los usuarios.

Los tres tipos básicos de recursos que necesitan ser gestionados son:

- Procesadores
- Memoria
- Dispositivos de E/S

La ***planificación de procesadores*** es crucial para el uso efectivo de los multiprocesadores. Las principales tareas de un planificador multiprocesador son:

1. Asignar los procesadores a las aplicaciones de manera consistente con los objetivos del diseño del sistema.

2. Asegurar el uso eficiente de los procesadores asignados a una aplicación.

El primero requisito afecta a la productividad del sistema y un segundo requisito afecta principalmente a la ganancia de velocidad. Dependiendo de la aplicación, la ganancia de velocidad o la productividad pueden tener prioridad. Estos dos objetivos son en cierto modo contrapuestos, ya que la máxima ganancia de velocidad puede exigir la dedicación de una gran porción de los procesadores del sistema a una sola aplicación. Por otro lado, la productividad puede mejorarse planificando varias aplicaciones para que sean ejecutadas conjuntamente, dedicando así menor número de procesadores a cada uno de ellas.

Debido a las dificultades de automatizar el proceso, la explotación del paralelismo dentro de una aplicación requiere generalmente un cierto grado de

orientación clara por parte del programador. La mayoría del soporte para el paralelismo explícito y automático los proporcionan los traductores de lenguajes y los procesadores. Puesto que el sistema operativo no es consciente de la semántica de los cálculos del usuario, generalmente no puede ofrecer gran ayuda. Sin embargo, el sistema operativo debería proveer mecanismos y facilidades que fomenten el paralelismo y complementen los esfuerzos de compiladores y programadores.

Los aspectos principales del soporte del sistema operativo para multiprocesamiento son:

1. Mecanismos de sincronización flexibles, eficientes entre procesadores y entre procesos.

2. Creación y gestión eficiente de un gran número de hebras de actividad, tales como procesos o hebras.

Este ultimo aspecto es importante ya que el paralelismo suele obtenerse dividiendo una aplicación en subtareas separadas individualmente ejecutables que pueden ser asignadas a diferentes procesadores.

La ***gestión de memoria*** en multiprocesadores es altamente dependiente de la arquitectura y del esquema de interconexión subyacentes. La memoria en sistemas débilmente acoplados se gestiona generalmente de forma independiente, de procesador a procesador.

En sistemas de memoria compartida, el sistema operativo debería proporcionar un modelo flexible de memoria que facilite al acceso seguro y eficiente a

estructuras de datos compartidas y a variables de sincronización. Un sistema operativo multiprocesador debería proporcionar un modelo unificado de la memoria compartida, independientemente del hardware, para facilitar el transporte de las aplicaciones entre diferentes entornos multiprocesadores.

La memoria compartida puede ser simulada en sistemas débilmente acoplados por medio de un mecanismo de paso de mensajes. Y a la inversa, el paso de mensajes puede ser eficientemente implementado en sistemas fuertemente acoplados apoyándose en la memoria físicamente compartida, proporciona un modelo flexible y un conjunto de herramientas para los programadores de aplicación.

La dualidad de gestión de memoria y comunicación entre procesos ha sido explotada por los diseñadores del sistema operativo Mach.

La gestión de dispositivos ha recibido poca atención en los sistemas multiprocesadores hasta la fecha. En parte se debe a que inicialmente la atención se ha centrado en la ganancia de velocidad para aplicaciones intensivas en cálculo que tienden a estar ejecutándose durante extensos períodos de tiempo y no generan demasiada E/S tras la carga inicial. Sin embargo conforme los multiprocesadores se apliquen aplicaciones más equilibradas de propósito general cabe esperar que sus necesidades de E/S aumenten en proporción con la productividad y la ganancia de velocidad conseguidas.

Es razonable esperar que la aplicación de técnicas de multiprocesamiento a la E/S, tales como encadenar una multitud de dispositivos de E/S y hacerlos funcionar en paralelo, produzca buenos resultados.

CAPITULO CUATRO

PROCESOS, CUESTIONES DE DISEÑO E IMPLEMENTACIÓN DEL SISTEMA OPERATIVO

4.1 LOS PROCESOS DESDE LA PERSPECTIVA DEL SISTEMA OPERATIVO

Un proceso se define como una instancia de un programa en ejecución. Desde el punto de vista de un sistema operativo, un proceso es la entidad más pequeña individualmente planificable, formada por código, datos y caracterizada por atributos y un estado dinámico. El código se compone de las instrucciones máquina y de las llamadas de servicio al sistema operativo. Los atributos asociados con un proceso son asignados por el programador del sistema o por el sistema operativo e incluye aspectos como la prioridad software y los derechos de acceso. El sistema operativo examina la ejecución de un proceso típico en el curso de su actividad en forma de progresión a través de una sucesión de estados. En la figura 4-1 se muestra una forma general del diagrama de transición de estados de los procesos.

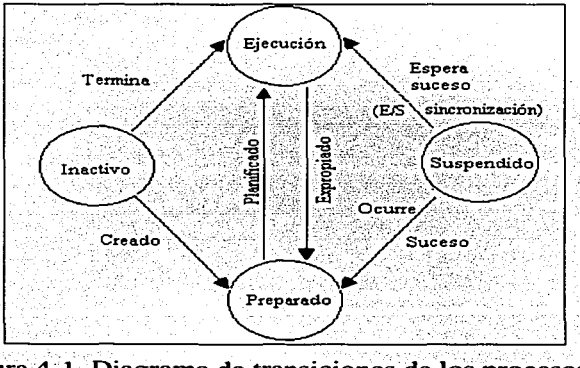


Figura 4-1 Diagrama de transiciones de los procesos [17]

[17] Véase www.aguila.itamx/bdd/unit.7htm

Un proceso creado esta en ejecución, listo para ejecutarse, o suspendido esperando un proceso. Existen cuatro categorías generales de los estados de un proceso que son:

- Inactivo
- Preparado
- En ejecución
- Suspendido

El estado ***inactivo***, es en cierta medida periférico, ya que se refiere a los procesos que aún no son conocidos, y por lo tanto no son registrados, por el sistema operativo. Todas las plantillas de proceso en espera de activación, además de los programas aún no remitidos al sistema operativo, pueden ser considerados como inactivos en esta clasificación.

Un proceso ***preparado***, tiene todos los recursos necesarios para su ejecución excepto el procesador. Los procesos asumen generalmente el estado preparado inmediatamente después de ser creados. Todos los procesos preparados están esperando que el sistema operativo les asigne el procesador para poder ejecutarse. Un módulo del sistema operativo denominado *planificador* selecciona uno de los procesos preparados para ser ejecutado cada vez que el sistema operativo toma el control del procesador y está en disposición de transferirlo a un proceso de usuario.

Un proceso ***en ejecución***, tiene los recursos necesarios para su ejecución incluyendo el procesador. En un sistema de un solo procesador, sólo puede

estar ejecutándose un proceso como máximo en cada instante. El proceso en ejecución ejecuta su secuencia de instrucciones máquina y puede invocar el sistema operativo para que efectué servicios tales como una operación de E/S o una sincronización mediante intercambio de señales en su nombre. Dependiendo del manejo de la planificación particular aplicada, el sistema operativo puede devolver el control al proceso en ejecución después de realizar el servicio, o puede planificar otro proceso si hay alguno preparado para ejecutarse.

Un proceso suspendido carece de algunos recursos además del procesador, por ejemplo de una señal de sincronización. Tales procesos están normalmente excluidos de la competencia por la ejecución hasta que desaparezca la condición de suspensión. El proceso en ejecución puede quedar suspendido al invocar una rutina de E/S cuyos resultados necesite para conseguir, o por la espera de una señal que aún no se haya producido. El sistema operativo registra entonces la razón de la suspensión de modo que pueda reanudar el proceso cuando la condición de suspensión desaparezca por efecto de las acciones de algunos otros procesos o por la llegada de un suceso externo.

Si el sistema operativo no vuelve al proceso en ejecución cuando una interrupción invoca al sistema operativo, el proceso de ejecución resulta efectivamente expropiado por un proceso de prioridad más elevada. Mientras tanto, el proceso expropiado permanece en el estado preparado en lugar del estado suspendido, ya que el único recurso de que carece es el procesador.

En cualquier momento, el conjunto de procesos activos asume diferentes estados. Los estados colectivos de todos los procesos y recursos (ocupados, libres) del sistema pueden ser considerados como el estado *global del sistema*.

En respuesta a sucesos externos e internos, los procesos pueden modificar rápidamente sus estados y formar otro estado global del sistema. Un sistema operativo multiprogramación lleva la cuenta continuamente de los cambios del estado global que ocurren como resultado de sucesos externos e internos. Utilizando el estado global del sistema como entrada, el sistema operativo emplea sus algoritmos de planificación para decidir cómo asignar recursos a los procesos solicitantes de tal modo que el rendimiento resultante del sistema satisfaga los objetivos de diseño.

El sistema operativo sigue la pista de todos los procesos, los suspende y los reanuda cuando resulta adecuado, asigna el procesador y otros recursos del sistema cuando es invocado para ello y proporciona una serie de servicios durante la ejecución de los procesos.

4.2 BLOQUE DE CONTROL DE PROCESO (BCP)

El sistema operativo agrupa toda la información que necesita conocer respecto a un proceso particular en una estructura de datos denominada descriptor de proceso o bloque de control de proceso (BCP). Cada vez que se crea un proceso (se inicializa, se instala), el sistema operativo crea un bloque de control de proceso correspondiente para que sirva como descripción en tiempo de ejecución durante toda la vida del proceso. Cuando el proceso termina, su BCP es liberado y devuelto al depósito de celdas libre del cual se extraen nuevos BCP. El estado inactivo se distingue de los otros estados porque un proceso inactivo no tiene BCP. Un proceso resulta conocido para el sistema operativo y por tanto elegible para competir por los recursos del sistema sólo cuando existe un BCP activo asociado con él. El bloque de control de proceso es una

estructura de datos con campos para registrar los diferentes aspectos de la ejecución del proceso y de la utilización de recursos. La información almacenada en un BCP incluye típicamente algunos o todos los campos siguientes:

1. Nombre del proceso (ID)
2. Prioridad
3. Estado (preparado, en ejecución, suspendido)
4. Estado hardware (registros y flags de procesador)
5. Información de planificación y estadísticos de uso
6. Información de gestión de memoria (registro, tablas)
7. Estado de E/S (dispositivos asignados, operaciones pendientes)
8. Información de gestión de archivos (archivos abiertos, derechos de acceso)
9. Información de mantenimiento

Algunos sistemas de multiprogramación incluyen información de mantenimiento con el propósito de facturar a los usuarios individuales, el tiempo de procesador, el almacenamiento, las operaciones de E/S y otras utilizaciones de recursos.

4.3 ESTADO DEL SISTEMA Y LISTAS DE PROCESOS

El estado de un proceso es sólo un componente del estado global del sistema, que engloba a todos los procesos y recursos. Para seguir la pista a todos los procesos, el sistema operativo mantiene listas de bloques de control de procesos clasificados por el estado actual de los procesos afectados. En general, existe una lista de procesos preparados, que contiene los BCP de todos los procesos preparados, y una lista de procesos suspendidos. En sistemas multiprocesadores puede haber una *lista de procesos en ejecución global*. Sin embargo, en sistemas monoprocesadores la lista de procesos en ejecución degenera a una sola entrada. Por tanto, generalmente es suficiente un solo puntero para identificar la *lista* de procesos en ejecución en un procesador único.

Mediante estas listas, el sistema operativo forma colecciones de procesos en estados análogos que probablemente son examinados por las rutinas de asignación de recursos del sistema operativo. Por ejemplo, el planificador buscara el siguiente proceso a ejecutar sólo en lista de preparados. El rendimiento del sistema operativo puede mejorar ordenando y actualizando estas listas de la manera más conveniente para las rutinas del sistema operativo que se saben que operan sobre ellas.

La figura 4-2 muestra las listas de sistema. Representa una instantánea del sistema multitarea, esta configuración en particular del BCP muestra el estado global del sistema, así como lo observa el planificador justo después de la primera pasada de RECOGER y antes de lanzar el proceso GUARDAR. Puesto que no hay ningún proceso de usuario en ejecución la lista de ejecución esta vacía. La lista de preparados consta de los BCP de los procesos

preparados, GUARDAR y CALCULAR en este caso. La de suspendidos contiene los BCP de IMPRIMIR y RECOGER, que quedaron suspendidos a la llegada de señales. En este ejemplo, supone que la lista de preparados está ordenada por las prioridades de los procesos con el fin de facilitar la planificación. La lista de suspendidos se gestiona en orden FIFO. La figura también indica el hecho de que los BCP contienen uno o más campos de enlace (punteros) para la formación de las listas.

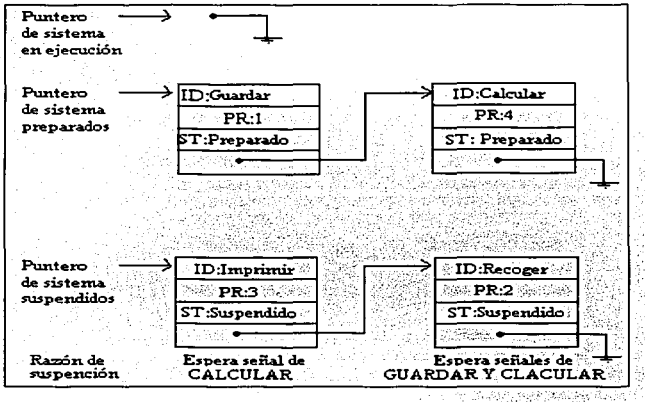


Figura 4-2 Listas del sistema: preparados y suspendidos [18]

4.4 CONMUTACIÓN DE PROCESOS

Una transición entre dos procesos residentes en memoria en un sistema de multiprogramación se denomina conmutación de proceso o conmutación de tarea. Puede ser bastante compleja y requiere una serie de pasos. Las

[18] Véase www.aguila.itamx/bdd/unit.7htm

principales operaciones implicadas en una conmutación de proceso se muestra en la figura 4-3.

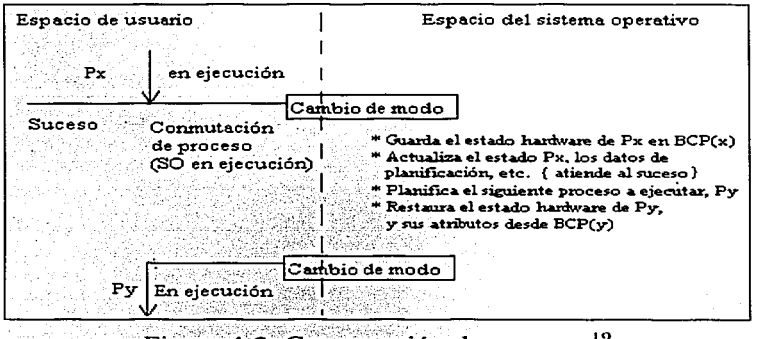


Figura 4-3 Conmutación de proceso [19]

Las conmutaciones de proceso ocurren generalmente en respuesta a sucesos que alteran el estado al sistema. El procesamiento de sucesos produce la transferencia de control al sistema operativo. Puesto que en un sistema operativo de multiprogramación se ejecuta generalmente en modo protegido y en un espacio de direcciones aparte del espacio de direcciones del usuario, la transición desde el proceso del usuario al sistema operativo requiere cruzar la frontera de protección. A esta operación se le denomina *conmutación de modo*.

El paso siguiente es guardar el estado del proceso en ejecución que está apunto de quedar suspendido o expropiado. El estado en tiempo de ejecución de un proceso incluye típicamente:

[19] Véase www.aguila.itamx/bdd/unit.7htm

- El contador de programa
- El puntero de pila
- La palabra del estado del procesador
- Los restantes registros accesibles del programa

A diferencia de cuando se guarda el contexto de interrupción, durante una conmutación de proceso hay que guardar el estado hardware completo. El estado hardware suele almacenarse en la pila de usuario, y el puntero del marco de pila afectado se guarda en el BCP del proceso saliente, designado como BCP (x). Otros campos relevantes de BCP(x) también pueden ser actualizados en ese momento. Candidatos potenciales son el estado, la razón de la expropiación o suspensión y una medida de uso del procesador. El BCP es trasladado a la lista de preparados o de suspendidos, dependiendo de que la causa de la conmutación de proceso haya sido una expropiación o una petición de un recurso que no pueda ser concedido motivo de suspensión.

El tratamiento del suceso que provocó el cambio del estado suele tener lugar en este momento, pero generalmente no se considera parte de la conmutación de proceso.

Tras la selección del siguiente proceso a ejecutar, su BCP es utilizado para preparar el adecuado entorno de ejecución. Además de restaurar el estado hardware, esta preparación puede incluir el cálculo de la máscara de interrupción asociada con la prioridad del proceso, el establecimiento de punteros para activar archivos y listas de recursos, la actualización de registros de gestión de memoria y otras acciones semejantes. Finalmente, el sistema

operativo inicia una conmutación de modo en el espacio del usuario y cede el control al proceso recién planificado.

La conmutación de proceso es:

- Es una operación bastante compleja
- Mas costosa que la conmutación de contexto de interrupción
- Puede ser bastantemente complicada en sistemas operativos grandes que disponen de detallado mantenimiento de recursos y sofisticados esquemas de planificación.

Dada su complejidad y relativa alta frecuencia de ocurrencia, la implementación de la conmutación de proceso puede afectar significativamente al rendimiento de un sistema operativo de multiprogramación. Es importante en sistemas orientados al rendimiento, como las aplicaciones de tiempo real.

La eficiencia de la conmutación de proceso puede ser mejorada con ayuda de asistencias de hardware y una técnica de estructuración de procesos especiales denominada *hebras*, que se menciona en la siguiente sección.

Un esquema hardware empleado para acelerar la conmutación de proceso es disponer de múltiples conjuntos estructuralmente idénticos de registros del procesador. Lo mínimo son dos:

1. Para el sistema operativo
2. Para los procesos de usuario.

Un bit dedicado en la palabra de estado del procesador indica el modo actual de operación, supervisor o usuario y el conjunto correspondiente de registros activos.

La idea de múltiples conjuntos de registros puede ser ampliada naturalmente en sistemas multiprocesadores proporcionando una multitud de procesadores, hasta el límite de uno por proceso. Tales sistemas pueden eliminar el problema de la conmutación de proceso, al menos el aumento aplicable a las expropiaciones, asignando un procesador dedicado a cada proceso activo.

4.5 HEBRAS DE EJECUCIÓN (THREADS)

Las hebras representan un método software para mejorar el rendimiento de los sistemas operativos reduciendo el recargo por conmutación de proceso. Una *hebra* es un proceso ligero con estado reducido. La reducción de estado se logra disponiendo que un grupo de hebras relacionadas compartan otros recursos tales como memorias y archivos. En sistemas basados en hebras, las hebras asumen el papel de los procesos como unidad individual más pequeña a efectos de planificación. En tales sistemas, el proceso o tarea sirve como entorno para la ejecución de hebras. El proceso se convierte por tanto en una unidad propietaria de recursos, tales como memoria y archivos, para una colección de hebras.

En sistemas con hebras, un proceso con una sola hebra es equivalente en un proceso clásico. Cada hebra pertenece exactamente a un solo proceso, y ninguna hebra puede existir fuera de un proceso. Los procesos son estáticos, y solo las hebras pueden ser planificadas para ejecución. Típicamente, cada

hebra representa un flujo separado de control y está caracterizada por su propia pila hardware. Como todos los recursos aparte del procesador son gestionados por el proceso que las engloba, la conmutación entre hebras relacionada es rápida y eficiente. Sin embargo, la conmutación entre hebras que pertenecen a diferentes procesos implica la conmutación de proceso completa, con su recargo asociado.

Las hebras son un mecanismo conveniente para explotar la concurrencia dentro de una aplicación. Las hebras pueden comunicarse eficientemente por medio de memoria compartida comúnmente accesible dentro del proceso que las engloba. Otras formas de mecanismos de comunicación y sincronización, tales como las señales, también pueden ser implementadas eficientemente debido a que las hebras relacionadas comparten memoria, y su interacción no requiere el recargo de cruzar fronteras de protección de memoria.

Las hebras representan un compromiso entre dos filosofías de implementación de los sistemas operativos que son:

1. Los tradicionales y pesados procesos que ofrecen protección pero que pueda afectar al rendimiento en tiempo real, tal como ocurre en UNÍX.

2. Los sistemas operativos de tiempo real ligeros y rápidos que sacrifican la protección y el rendimiento.

Las hebras proporcionan beneficios de velocidad y compartición para hebras relacionadas que constituyen una sola aplicación, mientras siguen proporcionando protección completa entre aplicaciones diferentes.

Las hebras han sido utilizadas con éxito en la implementación de servidores de red. También proporciona una base adecuada para la ejecución paralela de aplicaciones sobre multiprocesadores de memoria compartida. Con la sincronización entre hebras ya disponible el transporte de programas es fácil, y pueden obtenerse significativamente ganancias de velocidad de ejecución sobre un multiprocesador asignando procesadores separados a hebras individuales.

Las hebras son un desarrollo relativamente reciente en los sistemas operativos. Pueden hallarse en los sistemas operativos Mach y OS/2. varias implementaciones de paquetes de hebra como procesadores para lenguajes de programación, como C.

4.6 GESTIÓN Y PLANIFICACIÓN DE LOS PROCESADORES

Los aspectos principales de la gestión de los procesadores incluyen:

- Soporte para multiprocesamiento
- Asignación de recursos de procesamiento
- Planificación

4.6.1 SOPORTE PARA MULTIPROCESAMIENTO

El sistema operativo puede soportar multiprocesamiento y paralelismo facilitando un mecanismo para la creación y gestión de un gran número de procesos y/o hebras. El modelo de proceso puede considerarse que proporciona una abstracción de procesador virtual. Cada proceso tiene un

estado (contenido de los registros), un conjunto de registros asignados tales como archivos, memoria y conexiones con dispositivos de E/S. Una aplicación formada por varios procesos cooperativos representa un multiprocesador virtual que consta de procesadores virtuales y de los recursos que éstos comparten. La abstracción del multiprocesador virtual está a un nivel superior al del hardware, ya que incluye servicios del sistema operativo, tales como memoria virtual y sistemas de archivos.

Sobre monoprocesadores, la abstracción del multiprocesador se soporta multiplexando en el tiempo el procesador físico entre los procesadores virtuales. Se crea una ilusión de multiprocesamiento al asignar el procesador físico a un procesador virtual diferente durante cada intervalo de tiempo. En un entorno multiprocesador, es posible un multiprocesamiento real al asignar un procesador físico distinto a cada procesador virtual (proceso). El acceso a los recursos compartidos se mantiene generalmente a través de la memoria compartida o por medio de algún mecanismo de comunicación entre procesos.

En muchos sistemas operativos monoprocesadores, un proceso tiene un gran número de palabras dedicadas a su estado. Esto hace que la creación y conmutación de procesos sea bastante costosa. Además, las tablas de procesos que contienen información sobre el estado de cada proceso pueden consumir cantidades significativas de memoria cuando el número de procesos es grande. Las hebras son explicadas en la sección 4.4 para comprender este problema.

Cuando se utilizan hebras, cada aplicación se implementa como un proceso separado y sus partes concurrentes se codifican como hebras separadas dentro del proceso que las incluye. Las hebras pertenecientes a un mismo proceso comparten la memoria y todos los demás recursos adquiridos por el proceso.

Dada su estrecha relación y el objetivo común de ejecutar una aplicación, las hebras pertenecientes a un proceso tienen generalmente una interacción mucho más estrecha que las hebras que pertenecen a diferentes procesos. Algunos sistemas operativos facilitan llamadas especiales para sincronizar eficientemente hebras que pertenezcan a un mismo proceso. La comunicación no es normalmente una preocupación ya que las hebras comparten memoria común. Además de facilitar el multiprocesamiento y mejorar la eficiencia, las hebras pueden mejorar la planificación. Al tener información respecto a procesos y hebras, el planificador puede intentar coplanificar las hebras relacionadas y reducir así la probabilidad de retrasos provocados por la planificación desfasada de hebras estrechamente acopladas.

4.6.2 ASIGNACIÓN DE RECURSOS DE PROCESAMIENTO

La asignación del procesador puede ser un problema significativo en sistemas grandes paralelos. En este capítulo el termino proceso se refiere a una entidad planificable, que dependiendo de la implementación puede ser un proceso o una hebra. Una manera de seguir un gran número de procesos es organizarlos en una forma jerarquía lógica. La jerarquía lógica refleja organizaciones similares en el mundo real, como en grandes empresas o en la categoría militar.

Algunos procesadores son trabajadores y otros son gestores. Por lo tanto, la idea es dedicar a un gestor a llevar la cuenta del estado y la actividad de una colección de procesadores. Cuando es conveniente, el gestor recibe órdenes de trabajo procedentes de sus superiores y asigna el trabajo a las máquinas trabajadoras inactivas. La jerarquía puede extenderse hacia arriba asignando

gestores de segundo nivel que supervisen los gestores del primer nivel y así sucesivamente. Por fiabilidad, el nivel superior de una jerarquía puede estar formado por un grupo de nodos en vez de un solo nodo, cuya caída puede estropear el sistema.

El sistema denominado *planificación por oleadas*, en el sistema MICROS, que fue el fundador en la implementación de esta estrategia.

Los procesadores individuales se dedican a una tarea cada vez. Cada gestor lleva la cuenta del número de trabajadores que tiene disponible. Al recibir una petición de trabajo que solicita R procesadores, el sistema debe garantizar R o más procesadores, ya que alguno puede fallar. Las peticiones de trabajo pueden ser remitidas a un gestor en cualquier nivel de la jerarquía. Si el gestor elegido no tiene un número suficiente de trabajadores, pide ayuda a su jefe. Si el jefe no tiene suficiente ayuda disponible, puede llamar a su jefe a su vez. La operación continúa hasta que se obtiene un número suficiente de procesadores disponibles o alcanza la cima de la jerarquía. Si el intento de asignar a procesadores falla a ese nivel máximo, la petición puede retrasarse pendiente de la disponibilidad de una cantidad adecuada de recursos.

La asignación jerárquica de procesos es conceptualmente interesante ya que conduce a una implementación robusta y se escala bien debido a una cantidad relativamente limitada de tráfico generado. Es posible dotarla de tolerancia a fallos ya que el trabajo de un gestor que falla puede asignarse a uno de sus subordinados o a algún otro procesador designado por su jefe. Disponiendo de un comité en la parte superior del árbol jerárquico, el fallo de un miembro del comité puede ser paliado con la elección de un sucesor. El fallo de un proceso

trabajador no necesita paralizar el sistema si su trabajo puede ser reiniciado o reanudado por otro nodo.

La implementación de la planificación por oleadas crea algunas dificultades prácticas. Una de ellas es que un gestor puede tener una estimación obsoleta de la disponibilidad de su fuerza de trabajo. La necesidad del uso de estimaciones puede dar lugar a ineficiencias si son demasiado conservadoras, y a fallos de asignación si son demasiado optimistas. Además, puesto que múltiples peticiones de asignación pueden estar en proceso de ser concedidas al mismo tiempo, los recursos estimados como disponibles pueden ser captados por otro cliente en el momento en que se intente realmente una asignación del procesador.

4.6.3 PLANIFICACIÓN

El método jerárquico proporciona un modelo para el control de recursos en multiprocesadores y en sistemas distribuidos. Una vez que los procesadores son asignados a las aplicaciones de alguna manera, tienen que ser planificados.

Un objetivo deseable en los sistemas multiprocesadores es intentar coplanificar los procesos que interactúan de modo que se ejecuten al mismo tiempo. Los procesos situados a ambos extremos de un cauce, el emisor y el receptor de un mensaje, y quizás varias hebras pertenecientes a un solo proceso, son generalmente buenos candidatos para coplanificación. En caso contrario, se puede desaprovechar un tiempo considerable cuando se intenten intercambios entre entidades desfasadas.

En la figura 4-4 se muestra este problema. En el ejemplo, se planifican ocho procesadores para ser ejecutados en cuatro procesadores. Suponiendo que el sistema bajo consideración utiliza reparto de tiempos y planifican los procesos del grupo A, B, C, D en los intervalos de tiempo pares, y los procesos del grupo P, Q, R, S en los intervalos de tiempo impares. El procesador particular asignado a un proceso particular en un determinado intervalo de tiempo no importa para los propósitos de este ejemplo. Suponiendo que uno o más procesos del primer grupo se comunica (por ejemplo, mediante mensajes síncronos) con algunos procesos del segundo grupo.

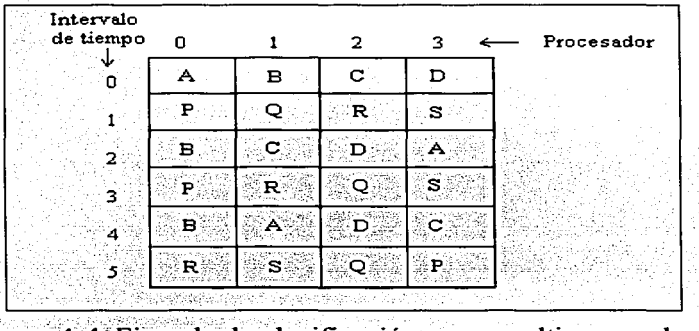


Figura 4-4 Ejemplo de planificación en un multiprocesador [20]

Por ejemplo, dentro de un intervalo de tiempo, el proceso A envía un mensaje al proceso P. Puesto que P no está ejecutándose, A queda bloqueado durante el resto del intervalo. Suponiendo que P recibe el mensaje al comienzo de su intervalo de tiempo y envía una replica inmediata, quedará bloqueado también hasta que A se ejecute otra vez. Si el intervalo de tiempo dura 50ms, la

[20] Véase www.aguila.itamx/bdd/unit.7htm

planificación que no toma en cuenta las relaciones entre los grupos de proceso podría restringir el intercambio de mensajes entre A y P como máximo uno cada 100ms.

Los procesadores individuales de un sistema multiprocesador pueden ser monoprogramados o multiprogramados. Un procesador puede estar dedicado a una sola tarea o puede multiplexar sus ciclos entre una serie de tareas que le sean asignadas.

La multiprogramación proporciona potencial para aumentar la productividad, pero puede agravar el problema de los retardos de comunicación por desfases. Un modo de abordar el problema es coplanificar grupos de procesos que se sabe o se cree que tienen comunicaciones entre sí. En la mayoria de los casos, la entidad iniciadora o el sistema operativo saben o pueden averiguar la identidad de la otra entidad. También deben facilitarse algunos medios para informar al planificador sobre la pertenencia a grupos y las relaciones entre ellos. Esta información puede ser determinista o servir como sugerencia de planificación.

Por ejemplo, el sistema operativo Mash soporta sugerencias concesoras por las cuales un proceso puede (conceder) control a otro. Esto puede ser útil, por ejemplo, para hacer que el emisor de un mensaje conceda control a su receptor.

Otro tipo de sugerencia es la *sugerencia desanimadora*, que puede ser útil para informar al planificador que un proceso prevé ciertos retrasos como consecuencia de su acción.

En un sistema débilmente acoplado, el planificador debería observar la afinidad de algunos procesos con ciertos procesadores. Éste puede ser el caso cuando el estado de un proceso esté almacenado en la memoria privada de un procesador específico.

El planificador debe considerar:

- La migración normalmente costosa en que se incurriría si reasignara tales procesos.
- El costo de las comunicaciones.

En general, el costo de las comunicaciones puede reducirse colocando procesos fuertemente interactivos entre sí sobre el mismo procesador o sobre una agrupación de procesadores que tengan enlaces mutuos directos. Estas consideraciones tienen que ser sostenidas frente al potencial del paralelismo, que generalmente aumenta cuando los procesos se asignan a diferentes procesadores.

4.6.4 GESTION DE MEMORIA

En multiprocesadores fuertemente acoplados, el sistema operativo debe facilitar y controlar el acceso a la memoria compartida. En tal entorno, el sistema operativo proporciona generalmente primitivas adicionales para asignación y desasignación de segmentos de memoria compartida. En sistemas que soportan memoria virtual compartida, varios búferes para apartado de traducciones (BAT) residentes en diferentes procesadores pueden contener traducciones de páginas que pertenecen a un segmento compartido.

En estos casos, el sistema operativo debe cooperar con el hardware para forzar la coherencia de los BAT. En caso contrario, tanto el acceso como la protección de la memoria pueden verse seriamente afectados. La coherencia BAT se soporta generalmente utilizando alguna variante de las técnicas desarrolladas para coherencias de caches en multiprocesadores.

Una vez que se dispone de soporte para memoria compartida, puede ser beneficioso extenderla al sistema de archivos. Esto se consigue utilizando primitivas existentes o añadiendo otras nuevas para proyectar los archivos sobre los espacios virtuales de los procesos. El resultado es un modelo de memoria uniforme disponible para los programadores y un mecanismo potencialmente eficiente para compartir archivos abiertos.

La memoria compartida puede ser también utilizada para mejorar el rendimiento del paso de mensajes. Se consigue evitando copiar los mensajes cuyos emisores y receptores tienen acceso a la misma memoria compartida física. En vez de ello, el sistema operativo puede transferir entre los dos procesos los derechos de acceso (y a veces tan sólo un puntero) al mensaje. El KMOS es un método que permite tanto al emisor como el receptor accedan al mensaje una vez completada la transferencia. Sin embargo, la modificación potencial del mensaje por parte de cualquiera de las entidades violaría la semántica normal del paso de mensajes que promete una copia separada para cada entidad.

La técnica de copiar en escritura puede ser utilizada para mejorar el problema mientras se mantiene una elevada eficiencia. Con este método, en la memoria compartida se guarda una sola copia del mensaje original. El paso de mensaje se realiza concediendo al receptor el derecho para acceder a esa copia. Sin

embargo, si alguna de las dos entidades intenta modificar una parte del mensaje, se genera un desvió hacia el sistema operativo. El sistema operativo se encarga entonces de crear una copia nueva de la pagina afectada, que es entonces asociada al proceso actualizador y que se utiliza para complementar la escritura. Por tanto, el emisor y el receptor continúan compartiendo las partes no modificadas del mensaje, y cada uno tiene una copia privada de las partes modificadas. Este método se beneficia del hecho de que la compartición de escrituras es relativamente infrecuente. Resguarda estrictamente la semántica del paso de mensajes mientras mantiene la eficiencia realizando una copia ociosa sólo cuando es inevitable, lo cual es posiblemente el caso para sólo una minoría de páginas. La copia en escritura ha sido aplicada también con éxito para reducir los costos de migración de procesos y pasos de mensajes en sistemas débilmente acoplados. En el sistema operativo Mach, se utiliza copia en escritura para la mayoría de los propósitos.

Algunos de los sistemas operativos para multiprocesadores débilmente acoplados proporciona la abstracción de memoria compartida y la implementación utilizando la facilidad de paso de mensajes.

CAPITULO CINCO

INTRODUCCIÓN A LA PROGRAMACIÓN PARALELA

Algunos sistemas multiprocesadores ofrecen herramientas automáticas, generalmente en compiladores o procesadores, para paralelizar los programas y aplicaciones de usuario. Si las herramientas no consiguen explotar el paralelismo potencial, o si el sistema no ofrece las herramientas, los programadores tienen que hacer la codificación para múltiples procesadores. Primeramente el programador comienza analizando el programa buscando recursiones y otras instrucciones iterativas que se ejecutan bastantemente a menudo a lo largo del programa Con frecuencia es útil manejar un perfilador para identificar dónde gasta el programa la mayor parte de su tiempo. Las iteraciones complejas más internas son generalmente un buen punto de partida. Después de realizar el análisis, el programador divide las regiones identificadas entre N procesadores, si es posible, y añade las instrucciones necesarias de sincronización y transmisión de datos.

5.1 GANANCIA DE VELOCIDAD

La ganancia de velocidad debido a la ejecución paralela de un programa está limitada por tres factores principales que son:

1. La proporción de secciones secuenciales frente a secciones paralelas en el código.
2. La ganancia de velocidad disponible para las partes paralelas, es decir, el menor valor entre el número de procesadores disponibles y el máximo grado de paralelismo codificado en el programa.
3. El recargo de preparación, sincronización y comunicación introducido por la operación paralela.

El último punto se refiere al recargo para preparar e inicializar las áreas compartidas de memoria, o para pasar datos a través de mensajes en sistemas débilmente acoplados, y a la creación de procesos para múltiples procesadores, a su comunicación y su sincronización en tiempo de ejecución.

La parte serie del programa debe ser ejecutada en serie, por lo que la ganancia de velocidad está limitada a las secciones paralelas. Gene Amdahl, estableció que la ganancia obtenida al introducir un componente más rápido está limitada por la fracción de tiempo que el componente más veloz está realmente en uso, específicamente.

$$G = \frac{1}{(1-f) + (f/k)}$$

Donde G es la ganancia ($G > 1$) relativa al caso estándar (serie) que resulta de la introducción de un componente k veces más rápido ($k > 1$) que es utilizado una fracción de tiempo f ($f \leq 1$).

La figura 5.1 muestra la gráfica de esta ecuación para diferentes valores de f, la fracción paralela de un programa. La ganancia de velocidad hardware aumenta hasta diez veces (para $k = 1$ a 10 procesadores) y se supone que es posible ejecutar en paralelo un número ilimitado de procesos preparados para su asignación a los procesadores.

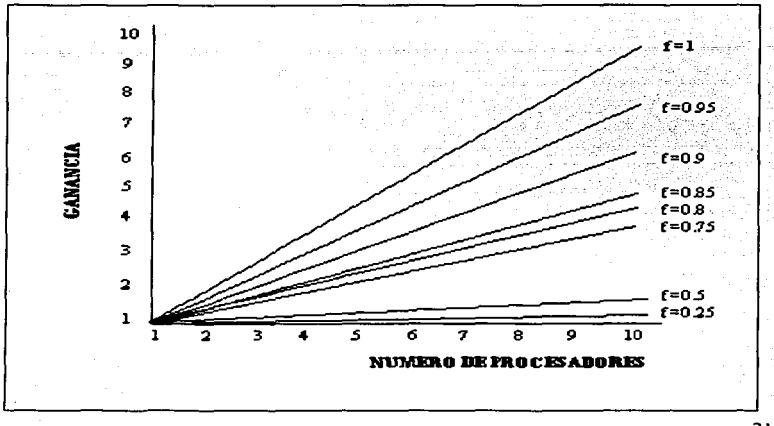


Figura 5-1 Ganancia de velocidad en función del paralelismo [21]

Una ganancia lineal ideal se consigue para programas que son totalmente paralelos hasta un grado que es menor o igual al número de procesadores disponibles en el sistema. Sin embargo, las partes serie del programa hacen que la máxima ganancia obtenible disminuya rápidamente. Por ejemplo, un programa que sea un 90 por 100 paralelo ($f = 0.9$) está limitado a una ganancia de aproximadamente 5.6 en un sistema con 10 procesadores.

[21] MILENKOVIC, Milan ,***Sistemas operativos conceptos y diseño***, 2ª ed., España, Editorial McGraw-Hill, 1994, p.595

5.2 UN EJEMPLO DE LA PROGRAMACION PARALELA: MULTIPLICACIÓN DE MATRICES

Como ejemplo de programación paralela, se considerara la multiplicación de dos matrices en un multiprocesador. El producto de dos matrices *A* y *B* n x n da lugar a una matriz *C* n x n donde los elementos pueden calcularse del modo siguiente:

$$C_{i,j} = \sum_{K=1}^{n} a_{i,k} \times b_{k,j}$$

Un forma de realizar la multiplicación matricial en paralelo es calcular cada fila (o cada columna) mediante un proceso separado. El programa 5.1 representa un modo de codificar la multiplicación matricial para un multiprocesador.

El programa contiene un proceso plantilla, *p_i*, para generar los procesos que calculan cada fila y que pueden ser ejecutados en procesadores distintos. El programa padre declara las tres matrices como variables globales que están colocadas en memoria compartida, accesibles a todos los procesadores.

En un sistema débilmente acoplado, el proceso padre tendrá que enviar los datos necesarios en forma de mensajes a los procesadores a los procesos en donde se van a ejecutar los procesos específicos de cálculo de fila.

El proceso padre inicializa el semáforo general HECHO con el número negativo de procesos trabajadores. Cada uno de ellos manda una señal al semáforo después de completar su cálculo. Cuando el semáforo recibe *N*

señales, queda libre. Este hecho es utilizado por el proceso padre para detectar el fin de la multiplicación matricial entera.

```
program mulmatriz; {multiplpicación paralela de matrices}
const N = . . . ; {n.º de filas}
var
i: integer; {privada, en el espacio del padre}
a: array [1. . N, 1. . N] of integer;
{variables locales compartidas}
b: array [1. . N, 1. . N] of integer;
c: array [1. . N, 1. . N] of integer;
hecho: semaforo; {general}
process p_i (i:integer); {se ejecutan N de estos procesos}
var
j, k, s: integer; {valores locales privadas}
begin
      for j := 1 to N do
      begin
      s := 0
      for j := 1 to N do s := s + a [i,k] * b [k,j];
      c[i,j] := s
      end; {for j}
      signal (hecho)
end; {proceso p_i}

begin {padre - inicialización}
      hecho := -(N); {se necesitan N señales para que quede libre}
      for i := 1 to N do initiate p_i(i);
      wait (hecho) {cuando se llega aquí, han finalizadolos N
procesos hijo}

end.
```

Programa 5.1 Multiplicación de matrices en un multiprocesador [22]

[22] MILENKOVIC, Milan ,***Sistemas operativos conceptos y diseño***, 2ª ed., España, Editorial McGraw-Hill, 1994, p.596

Después de inicializar el semáforo de terminación, el proceso padre inicia los procesos de cálculo de filas. Se supone que cada uno de ellos recibe como parámetros, *i*, un número de fila diferente. Así, las filas se calculan en paralelo en procesos separados. El paralelismo obtenido depende del número de procesos disponibles y del grado del paralelismo de la aplicación es n, ya que no se ha efectuado subdivisión por debajo del nivel de fila.

La figura 5.2 muestra un diagrama de tiempos de un escenario de ejecución del programa sobre un multiprocesador de memoria compartida con $N = 12$ y cinco procesadores.

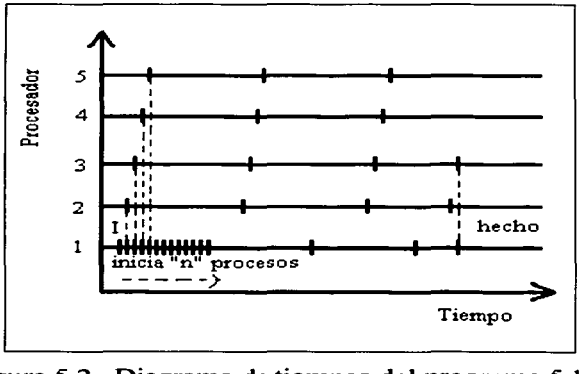


Figura 5.2 Diagrama de tiempos del programa 5.1 [23]

[23] MILENKOVIC, Milan ,***Sistemas operativos conceptos y diseño***, 2ª ed., España, Editorial McGraw-Hill, 1994, p.597

El diagrama verifica que el procesador 1 ejecuta la tarea padre, la cual inicializa el sistema y la memoria compartida (designada como I en la figura 5.2) y luego inicia los 12 procesadores de computación de filas. Los 4 primeros procesos son asignados inmediatamente a los procesadores libres, y los 8 restantes son colocados en la lista de preparados. Conforme los procesadores quedan disponibles los procesos son planificados para su ejecución. Se verifica que los procesadores estén multiprogramados. Por consecuencia, el procesador 1 comienza a ejecutar el siguiente proceso trabajador (5) después de que el proceso padre queda suspendido pendiente de la apertura del semáforo HECHO.

No es necesaria más sincronización que la del semáforo de terminación en el ejemplo de la multiplicación matricial tal como se ha codificado en el programa 5.1. Esto se debe a que los procesos separados comparten las dos matrices multiplicados en modo de sólo lectura, y cada una de ellas escribe una fila distinta de la matriz resultado C.

Dependiendo de la planificación de los procesos, de la organización de la memoria y de la disposición de los datos, pueden existir algunas colisiones al nivel del hardware cuando diferentes procesadores intenten acceder a las mismas celdas de memoria simultáneamente. Como indica la figura 5.2, los procesos son iniciados en varios procesadores en momentos diferentes. Si las matrices están colocadas en la memoria compartida en una secuencia similar al orden en el que el programa las procesa, no es muy probable que varios procesadores entren en sincronismo y colisionen cuando accedan a la memoria compartida.

Como precaución, el producto parcial, *s*, está declarado como variable local privada para cada proceso. Consecutivamente , los productos intermedios podrían ser almacenados directamente en la copia global de la matriz C a expensas de la contención de memoria. De igual modo, la contención se puede empeorar si los datos que se procesan en un orden que esté en desacuerdo con la secuencia de almacenamiento de las matrices en la memoria producida por el compilador.

En general, la organización del programa y de la disposición de los datos es muy importante para el rendimiento de los programas paralelos. Una buena estructuración puede beneficiar al tiempo de ejecución, y una ejecución puede hacer descender la velocidad de un programa en teoría altamente paralelo al aumentar la probabilidad de contención sobre la memoria y sobre las rutas de interconexión en tiempo de ejecución.

5.3 FORK (DIVIDIR) Y JOIN (UNIR) EN MULTIPOCESADORES

Conway propuso dos primitivas para la creación y terminación de grupos de procesos en multiprocesadores, FORK (DIVIDIR) y JOIN (UNIR). La instrucción FORK-DIRECCIÓN genera un nuevo proceso que comparte toda o parte de la memoria con el proceso que ejecuta FORK. El proceso que ejecuta FORK continúa ejecutándose sobre el mismo procesador. El proceso dividido es asignado a otro procesador si hay alguno disponible; en caso contrario queda depositado en una cola esperando ejecución.

La instrucción JOIN tiene el propósito de reunir los flujos de control generados por FORK después de la terminación de su trabajo. Generalmente se asocia a

un JOIN un contador de números de procesos. Al terminar su operación, cada proceso ejecuta una instrucción JOIN *N*. Esto hace que el contador se incremente y se compare con *N*. Si el resultado es menor que *N*, el trabajo no ha acabado, y hay mas procesos que deben terminar. Además, la ejecución de JOIN termina efectivamente el proceso que la emite y sus recursos pueden ser liberados y devueltos al deposito de asignaciones. El último proceso generado finaliza y ejecuta JOIN *N*. Esto devuelve el control al proceso que emitió la instrucción FORK.

Las instrucciones FORK y JOIN pueden ser utilizadas para implementar la aplicación del programa 5.1 sin utilizar semáforos. Esto puede lograrse haciendo que el proceso de control (padre) genere *n*-1 procesos idénticos de cálculo de filas y les pase un valor diferente de *n* a cada uno. El cálculo de la enésima fila puede ser efectuado en el cuerpo del propio proceso de control. Cada uno de los procesos generados terminaría con una instrucción JOIN 12. Cuando hayan ejecutado JOIN 12, se habrá acabado la multiplicación de matrices, y el proceso de control puede proseguir con otras actividades o terminar. Esto modificaría el diagrama de ejecución de la figura 5.2 al ejecutar el proceso 12 sobre el procesador 1 quedaría libre para otras asignaciones hasta terminar JOIN.

Las instrucciones FORK y JOIN están ajustadas a las necesidades de la programación de multiprocesadores. Los procesos generados mantienen una estrecha relación y acceden a datos compartidos en memoria. Las instrucciones JOIN proporcionan un modo de concluir el cálculo y señalar la obtención del resultado sin tener que recurrir al uso de otros mecanismos de sincronización, tales como semáforos.

CAPITULO SEIS

SINCRONIZACIÓN EN MULTIPROCESADORES

La sincronización y comunicación entre procesadores son necesarias para diseñar software correcto y fiable. La ejecución paralela de programas y la compartición de lectura/escritura de los datos establece requerimientos sobre los mecanismos de sincronización en multiprocesadores.

En multiprocesadores débilmente acoplados tanto la sincronización como la comunicación entre procesos se manejan mediante mensajes. Se transmiten los datos necesarios a los procesadores individuales para su procesamiento, existe generalmente poca necesidad de que los procesos se sincronicen mientras se operan sobre los datos en memoria local. Puede ser necesario sincronizar para comunicar los resultados y para consolidarlos en cualquiera que sea el nodo que se vaya a guardar después de terminar el cálculo paralelo.

El acceso a memoria compartida puede mejorar en la velocidad para muchas aplicaciones, pero también aumenta la necesidad de sincronización. Algunas instrucciones de monoprocesador son diseñadas, tales como *comprobar-y-fijar* y *comparar-e-intercambiar* implementadas utilizando el ciclo de lectura-modificación-escritura, pueden ser utilizadas como base para la sincronización entre procesos en sistemas multiprocesadores.

6.1 ¿QUÉ ES UN SEMÁFORO?

Un mecanismo semáforo consta de dos operaciones, señal (SIGNAL) y espera (WAIT), que operan sobre un tipo especial de variable semáforo, *s*. La variable semáforo puede tomar valores enteros y, excepto posiblemente en su inicialización, sólo puede ser accedida y manipulada por medio de las

operaciones SIGNAL y WAIT, ambas tienen un argumento cada una, la variable semáforo, y se definen de la siguiente manera:

- WAIT(S), decrementa el valor de su argumento semáforo, *s*, mientras el resultado no sea negativo, debe ser indivisible.

- SIGNAL(S), incrementa el valor de su argumento semáforo, *s*, en una operación indivisible.

Un *semáforo binario* tiene permitido tomar los valores 0 (ocupado) y 1 (libre), la lógica del wait(s) debe interpretarse como la espera hasta que la variable semáforo *s* sea igual a LIBRE, después de su modificación para que indique OCUPADO antes de devolver el control al invocador. La operación WAIT implementa la fase del protocolo de exclusión mutua. SIGNAL pone el valor de la variable semáforo LIBRE y representa la fase de liberación de exclusión mutua.

Un *semáforo general* puede tomar cualquier valor entero. La lógica de las operaciones WAIT y SIGNAL. Cuando están disponibles, las operaciones de semáforo y las declaraciones de variables semáforo se proporcionan generalmente en forma de llamadas al sistema operativo o como funciones y tipos incorporados a un lenguaje de implementación de sistemas.

6.1.1 PROPIEDADES DE LOS SEMÁFOROS

Los semáforos son un mecanismo sencillo pero poderoso de asegurar la exclusión mutua entre procesos concurrentes para acceder a un recurso

compartido. Cuando se utilizan semáforos, las modificaciones de código o la reestructuración de procesos y módulos no señalan cambios en otros procesos, e incluso si pertenecen a la misma familia y comparten los mismos recursos.

Los semáforos pueden estar disponibles en un lenguaje de programación, como construcción de lenguaje, o como servicio del sistema operativo invocado mediante llamadas al sistema. Cuando son proporcionadas por el sistema operativo, las variables semáforo no son declaradas ni manipuladas en el lenguaje, si no que manipulan a través de llamadas al sistema tales como CREAR_SEMÁFORO, ASOCIAR_A_SEMÁFORO, ESPERA, SEÑAL y CERRAR_SEMÁFORO.

6.2 COMPROBAR-Y-FIJAR

La instrucción comprobar-y-fijar (TS, *test-and-set*) está diseñada para resolver conflictos entre procesos haciendo posible que sólo un proceso reciba permiso para entrar a su sección crítica. Cada proceso que desee acceder al recurso debe obtener un permiso para esto se ejecuta la instrucción TS con la variable de control asociada como operando. Cuando hay varios procesos concurrentes compitiendo, la instrucción TS asegura que sólo uno de ellos pueda utilizar el recurso.

TS lleva un operando, la dirección de la variable de control o de un registro que pueda actuar como semáforo, estos pasos se realizan en una única operación y funciona del modo siguiente:

TS *operando*: ;comprobar y fijar

1. Comparar el valor del operando con OCUPADO y modificar los códigos de condición para que reflejen el resultado.

2. Fijar el operando a OCUPADO.

Un modo de implementar un semáforo en un sistema multiprocesador es declararlo como variable compartida y actualizarlo como sea necesario según la definición de semáforo con la ayuda de instrucciones comprobar-y-fijar. En estos sistemas, varios procesadores pueden esperar a que el semáforo se abra ejecutando instrucciones TS en paralelo. La operación de semáforo WAIT (esperar) puede implementarse utilizando:

- Espera activa (*spin lock*) o

- Cola de procesos en espera.

Desafortunadamente, ninguna de las dos alternativas son atractivas en sistemas multiprocesadores. La espera activa, denominada *spin lock* en multiprocesadores desaprovecha ciclos de procesador y consume el ancho de banda de los enlaces que conectan los procesadores a la memoria compartida. Además, varios procesadores que consulten continuamente un cierre de semáforos pueden provocar detención del módulo de memoria que contiene la variable semáforo y perjudicar así el acceso por parte de otros procesadores al banco de memoria que la incluye.

En sistemas multiprocesador con múltiples caches como muestra la figura 6.1, las consultas continuas de cierres pueden provocar incrementos en el trafico de bus necesario para mantener la consistencia entre las copias de la variable semáforo que residen en las caches de los procesos competidores. Dependiendo del esquema de coherencia de caché aplicado, pueden presentarse problemas adicionales relacionados con las caches y degradación del rendimiento.

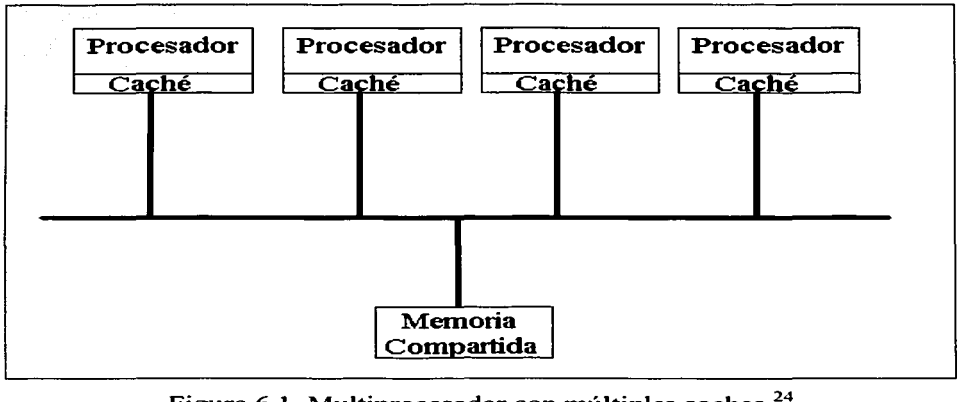


Figura 6.1 Multiprocesador con múltiples caches [24]

Las colas de procesos en espera pueden disminuir alguno de estos problemas, tanto la inserción en cola como la posterior extracción de cola y la activación de los procesos suponen una serie de operaciones. Pero algunas de esas

[24] MILENKOVIC, Milan ,***Sistemas operativos conceptos y diseño***, 2ª ed., España, Editorial McGraw-Hill, 1994, p.578

operaciones acceden simplemente a estructuras de datos compartidas (como la propia cola de semáforos) y deben ser protegidas mediante cierres.

6.3 COMPARAR-E-INTERCAMBIAR

La instrucción comparar-e-intercambiar (CS, *compare-and-swap*) sigue una estrategia para resolver el problema de la exclusión mutua. La instrucción CS es muy conveniente y eficaz para el caso de actualizaciones simples de variables compartidas. Proporciona una base para el control de concurrencia que no utiliza bloqueo por cerraduras. Si se implementa utilizando el ciclo de memoria indivisible de lectura-modificación-escritura, la instrucción CS puede ser utilizada en multiprocesadores. Uno de sus usos es la introducción y extracción de una cola compartida.

La gestión de las listas y colas compartidas es una tarea importante que es frecuentemente realizada en sistemas operativos multiprocesadores. Una técnica de planificación frecuente en sistemas de memoria compartida es mantener una lista de procesos preparados en memoria compartida para ser consumidos por los procesadores inactivos.

Otras instancias de listas y colas compartidas son las colas de semáforos, las colas de recursos compartidos y las colas de buzones.

En el código de KMOS pueden encontrarse en varios tipos de operaciones de gestión de colas y listas. Evidentemente, el acceso a colas compartidas debe estar controlado para evitar manipulaciones inconsistentes por parte de actividades concurrentes iniciadas por diferentes procesadores. La instrucción

comparar-e-intercambiar proporciona un vehículo para la manipulación de colas sin bloqueos por cierres.

La figura 6.2 muestra un ejemplo del aspecto de una cola en forma de lista simplemente enlazada. Dos punteros, *Primero y Último*, apuntan al primero y al último elemento de la lista, respectivamente.

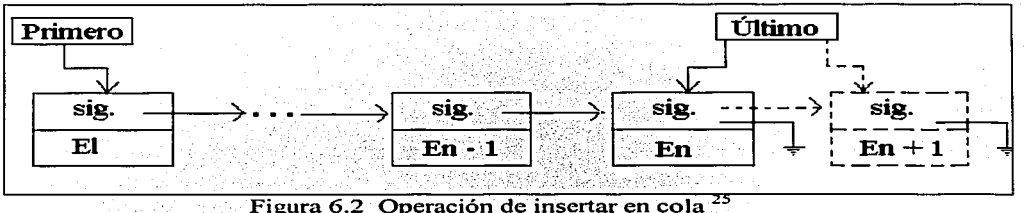


Figura 6.2 Operación de insertar en cola [25]

Suponiendo que los elementos se insertan por el final y se extraen por el principio, las líneas punteadas de la figura 6.2 muestran el aspecto de la lista después de añadir un nuevo elemento, E_{n+1}. En notación de KMOS y Pascal, un nuevo elemento ***elemento*** puede ser añadido utilizando las siguientes instrucciones:

```
Elemento ^.siguiente: =nil
temp: Último;
Último:= elemento; {actualiza Último para que apunte al elemento insertado}
```

[25] MILENKOVIC, Milan ,***Sistemas operativos conceptos y diseño***, 2ª ed., España, Editorial McGraw-Hill, 1994, p.600

temp^.siguiente: = elemento; {el enlace siguiente del anteriormente último elemento apunta a la nueva entrada}

Elemento, Último y *temp* denotan punteros a las variables correspondientes. La notación *puntero^.siguiente* se refiere al campo *siguiente* (enlace) del nodo al que puntero apunta en la lista.

Cuando varios procesadores intentan añadir diferentes elementos a la misma lista concurrentemente, se pueden producir percances tales como la pérdida de una actualización o el entrecruzamiento de punteros cuando las instrucciones precedentes se ejecutan en dos o más en modo entrelazado. Una manera de evitar esto es tratar la operación de añadir (insertar la cola) como sección crítica. La instrucción comparar-e-intercambiar puede ser utilizada para añadir una entrada, elemento, sin el recargo de bloquear con cierre e invocar al SO, del modo siguiente:

```
        elemento^.siguiente := nil;
        último_anterior:= Último;
BUCLE:  CS último_anterior, elemento, Último;
        if (último_anterior no era = Último)
        the goto BUCLE; {si no Último = elemento}
        último_anterior^.siguiente:= elemento;
```

El nuevo elemento se convierte en la última entrada de la lista por lo que su enlace *siguiente* toma el valor nil. El valor actual del puntero *Último* se copia en el registro ANTERIOR de CS, designado como *último_anterior* en el código. Después se utiliza comparar-e-intercambiar para tratar de depositar el elemento en la cola haciendo que *Último* apunte al nuevo elemento. La

instrucción CS se utiliza para verificar que no se realicen inserciones en la lista por parte de otros procesadores mientras tanto. Si esto ocurriera así, CS fallaría y devolvería el valor nuevo, actualizado, de *Último* en el registro *último_anterior*. Posiblemente, la actualización tendrá éxito, y la variable global *Último* apuntará al elemento. En ese momento, el enlace *siguiente* del elemento inmediatamente precedente en la lista (el elemento anteriormente último) apuntará a la nueva entrada.

La operación básica de retirar (extraer de cola), suponiendo que la lista no está vacía, puede implementarse con compara-e-intercambiar del modo siguiente:

```
        primero_anterior:= primero;
BUCLE:  if primero_anterior <> nil
          then
            begin
                    elemento:= Primero_anterior□.siguiente;
                    CS primero_anterior, elemento, Primero;
                    if (primero_anterior no era = Primero) the goto
        BUCLE
            end
          else... {la lista está vacíia}.
```

Si la lista está vacía ocasiona complicaciones adicionales y requiere un código más elaborado. Estos problemas pueden evitarse manteniendo un elemento ficticio de tal modo que la lista nunca esté vacía. Esta técnica ha sido utilizada en el código KMOS, por ejemplo, para evitar vaciar la lista de preparados.

Puede demostrarse que la primera secuencia de código presentada funciona correctamente en presencia de operaciones concurrentes de inserción en cola. La secuencia de extracción de cola funciona correctamente para extracciones concurrentes siempre que la lista no esté vacía. Sin embargo, la solución comparar-e-intercambiar presentada no es correcta cuando se permiten operaciones de insertar y extraer de cola simultáneas sobre la misma cola compartida. Por ejemplo, la operación de insertar en cola supone que la igualdad de las variables *último_anterior* y *Último* implica que no han sido añadidos elementos a la cola de momento. Con inserciones y extracciones de cola concurrente puede que esto no sea cierto. Es posible, que dos procesos estén entrelazados de tal modo que extraiga un elemento y luego se inserte inmediatamente en la cola. Esto dejaría el puntero *Último* sin modificar y confundiría a la secuencia de insertar en progreso de tal modo que dos procesadores pueden efectuar una actualización errónea al depositar dos valores diferentes en la misma posición de memoria, *úntimo_anterior^.siguiente*, sin sincronización.

Este fallo es ocasionado por la incapacidad de comparar-e-intercambiar de detectar una modificaciones. Una instrucción más potente, *comparar-doble-e-intercambiar* reduce el problema de ampliar la operación normal de comparar-e-intercambiar para que incluya una comparación del contador de modificaciones. Ambas comparaciones (los valores de referencia nuevo, antiguo y el contador nuevo, antiguo) deben coincidir para que comparar-doble-e-intercambiar tenga éxito.

La instrucción comparar-e-intercambiar funciona en los multiprocesadores cuando se implementa utilizando el ciclo indivisible de lectura-modificación-escritura en memoria compartida. La instrucción ayuda a la implementación

del control de concurrencia optimista para algunas aplicaciones, como la actualización de variables compartidas y la manipulación de lista compartidas.

Lo importante del método CS para control de concurrencia es la ausencia de bloqueo por cierres, donde garantiza que al menos uno de los procesos podrá estar progresando con independencias con sus prioridades y orden de planificación.

La aplicación directa de la instrucción CS resuelve una clase de problemas de sincronización sin imponer el recargo de las llamadas al sistema operativo. En el aspecto negativo, la instrucción comparar e intercambiar es un poco complicada y difícil de usar correctamente.

6.4 ACCEDER-Y-SUMAR

El problema de rendimiento usual con el ciclo de memoria de lectura-modificación-escritura es la complejidad y la contención de memoria provocada por los protocolos de bloqueo. Si *N* procesos están tratando de acceder a una sección crítica al mismo tiempo, el sistema de memoria debe de ejecutar N operaciones básicas de cierre, una tras otra, aun cuado máximo sólo un proceso puede llegar a tener éxito.

La instrucción acceder-y-sumar (FA, *fetch-and-add*) es ejecutada por la lógica situada en la red de conmutación de interconexiones que conecta los procesadores con la memoria. La primitiva acceder-y-sumar se denota como:

FA (x, i)

donde

x es una variable en memoria compartida

i es incremento que se le suma cuando se completa la instrucción.

Cuando solo un procesador solicita la ejecución de acceder-y-sumar sobre *x*, la semántica es:

```
acceder-y-sumar (x,i) {definición}
        begin
          temp:= x;
          x : = temp + i
          return  Temp.
        end
```

Si *N* procesadores intentan ejecutar acceder-y-sumar simultáneamente sobre la misma posición de memoria, los incrementos se suman y se combinan en la red de conmutación de manera que la posición de memoria *x* se le aplica un solo incremento que es la suma de los *N* incrementos. En respuesta a esta instrucción, se devuelve un único valor a cada procesador.

El principal beneficio de acumular los incrementos en la red es reducir la contención, ya que los resultados pueden ser devueltos a los procesos originarios en paralelo por los elementos de conmutación contiguos, y la posición de memoria global sólo necesita ser accedida una vez para aplicar el resultado acumulado final. Los valores individuales devueltos corresponden a una serialización arbitraria de la *N* peticiones.

Desde la parte del procesador y de la memoria el efecto es idéntico a una ejecución en serie de las *N* instrucciones acceder-y-sumar. Dando lugar a estas instrucciones son realmente realizadas mediante una actualización de memoria de un sólo ciclo, la primitiva acceder-y-sumar puede ser muy efectiva para acceder a las colas compartidas y para la gestión de procesos idénticos que operan sobre diferentes segmentos de datos.

Por ejemplo, el programa de multiplicación de matrices en el programa 5.1 extiende el trabajo entre muchos procesadores, donde cada uno de los cuales calcula una fila diferente. La parte serie del programa se inicializa y distribuye un índice de fila diferente para cada proceso. Con la instrucción acceder-y-sumar, el mismo efecto podría lograrse haciendo que cada procesador ejecutara el siguiente código para una matriz con *N* filas:

```
n :=FA (R, 1)
while (n <= N) do
  begin
     {cálculo de la fila n};
     n := FA (R, 1)
end; {while}
```

Cada procesador ejecuta inicialmente acceder-y-sumar y obtiene un índice de fila diferente.

Donde:

n es utilizado en el segmento de código

R es una variable global que acumula el número de filas calculando y que toma el valor 1 inicialmente.

En la secuencia de código presentada, cada procesador ejecuta la instrucción FA (R, 1), obtiene un valor diferente de n, por lo que calcula una fila diferente. Al terminar su cálculo, el procesador ejecuta otra instrucción FA para obtener el número de la fila siguiente que necesita ser calculada. El primer procesador comprueba el valor obtenido contrastándolo con el tamaño de la matriz, *N*. Si es menor que la constante *N*, el cálculo ha terminado y el procesador sale del while. Ya que la ejecución paralela de la instrucción acceder-y-sumar devuelve a cada procesador un valor único de R, cada uno de ellos de ellos calcula una fila diferente de la matriz.

La comparación con *N* asegura que la solución funciona para un número arbitrario de procesadores que puede ser menor o mayor que *N*. Este esquema sólo consigue la distribución de los índices de fila y que también permite que los procesadores se planifiquen a sí mismos sin necesidad de utilizar una cola o una lista compartida de procesos preparados.

CAPITULO SIETE

SISTEMAS MULTIPROCESADORES COMERCIALES

Los sistemas multiprocesadores comerciales se encuentran disponibles en el mercado y la mayoría de los multiprocesadores comerciales que existen funcionan con operaciones SISD múltiples al más alto nivel del programa y ofrecen un acoplamiento ligero, sin embargo, el grado de acoplamiento varía de sistema a sistema. Los sistemas operativos multiprocesadores consiguen la concurrencia a través del paralelismo.

7.1 IBM 370/168MP

El multiprocesamiento es una característica en las series para la línea de alto rendimiento. La mayoría de los modelos del sistema 370 son maquinas SISD con un monoprocesador.

En la figura 7.1 se muestra una configuración IBM 370/168 monoprocesador.

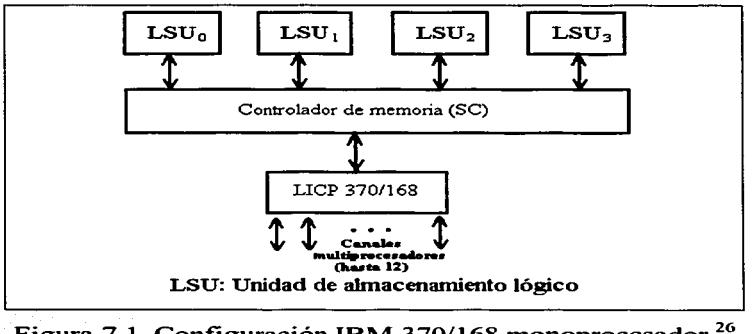
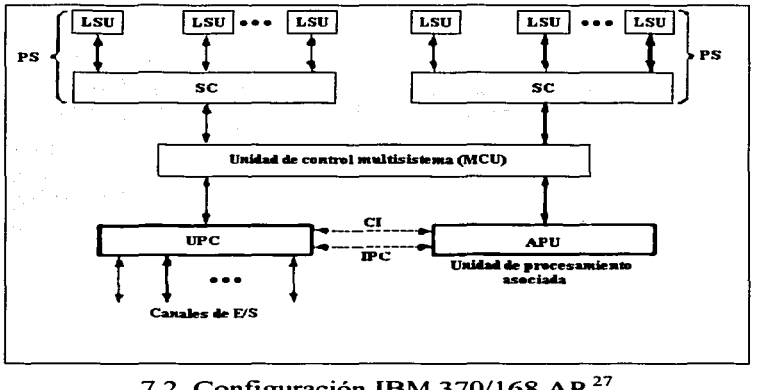


Figura 7.1 Configuración IBM 370/168 monoprocesador [26]

[26] HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, p.745

Donde la memoria está dividida en cuatro unidades de almacenamiento lógico (LSU, *Logical Storage Units*) y forman un sistema de memoria entrelazada de cuatro vías. Los accesos a memoria y la resolución de conflictos están regulados por el controlador de memoria. Existe únicamente una unidad de procesamiento central (UCP) que comprende las unidades de segmentación dirigidas para decodificar y ejecutar instrucciones junto con una caché rápida.

Pueden conectarse múltiples canales de E/S a la UCP y cada canal es un procesador de E/S con características de un conjunto de instrucciones de E/S simple, y opera asincrónicamente con la UCP. La *configuración AP* de IBM 370/168 se muestra en la figura 7.2, es la extensión de la configuración del monoprocesador que incluye la unidad de procesamiento asociada (APU).



7.2 Configuración IBM 370/168 AP [27]

[27] HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, p.746

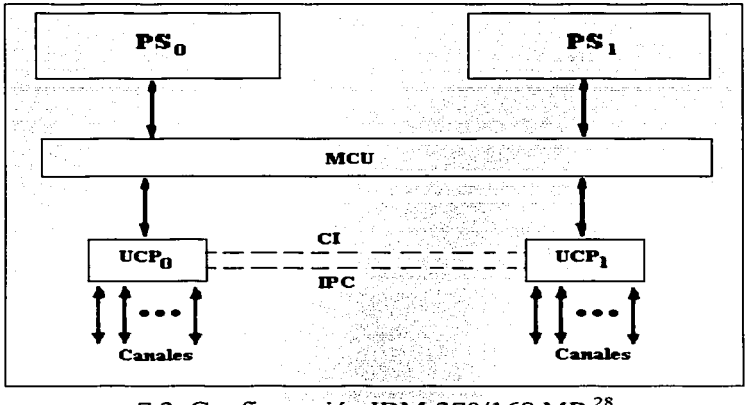
La APU es semejante a la UCP 370/168 estándar, la diferencia que existe es que no se pueden asociar canales de E/S, sin embargo, tienen sus propias caches. El inconveniente de la coherencia caché se soluciona utilizando líneas de invalidación caché (CI) entre la UCP y APU. Las líneas de comunicación interprocesador (IPC) se utilizan para intercambiar información o señales de interrupción dirigidas entre los dos procesadores.

La estructura no es igual porque:

- La APU se dedica especialmente a la computación.
- La UCP maneja tanto la computación como las comunicaciones.

La unidad de control multisistema (MCU) efectúa las funciones de conmutación de interconexión entre los procesadores y los módulos de memoria compartida.

En la figura 7.3 muestra una *configuración MP de IBM 370/168*, ahora se utiliza otra UCP para formar un sistema biprocesador simétrico. La configuración está compuesta por dos sistemas monoprocesadores 370/168 con memorias compartidas. Las dos UCP son de igual capacidad y dos conjuntos de canales de E/S son asociados a cada UCP y son mutuamente exclusivos pero no pueden comunicarse directamente. La MCU proporciona el hardware de interconexión entre las dos UCP y las memorias compartidas.

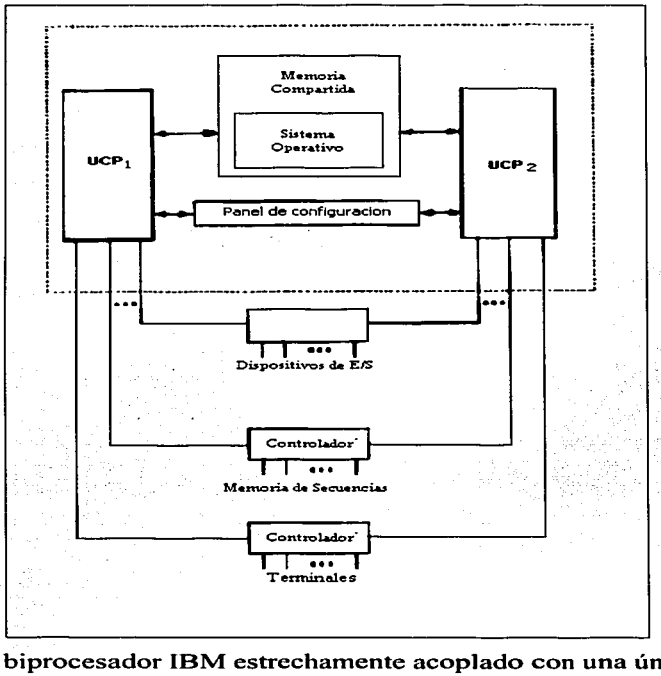


7.3 Configuración IBM 370/168 MP [28]

La configuración 370/168 MP se considera ligeramente acoplada porque ejecutan dos copias separadas del sistema operativo en las dos UCP. Un sistema monoprocesador 370/168 puede ser reconfigurado como un sistema multiprocesador estrechamente acoplado de dos UCP con memorias compartidas y dispositivos de E/S compartidos, en la figura 7.4 se muestran las dos UCP que están estrechamente acopladas mediante una única copia del sistema operativo en la memoria compartida. Un par de UCP estrechamente acopladas también pueden estar ligeramente acopladas con otra UCP monoprocesadora para formar un multiprocesador mixto como se muestra en la figura 7.5, es un multiprocesador estrechamente acoplado en una configuración ligeramente acoplada.

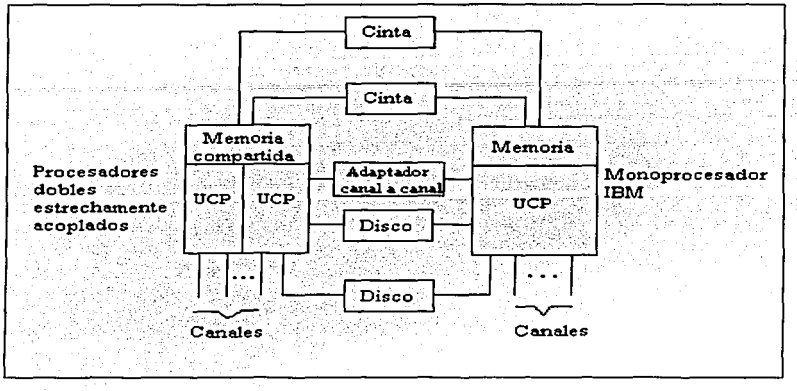
[28] HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, p.746

La doble UCP estrechamente acoplado y el monoprocesador comparten algunos dispositivos de acceso directo, como discos y algunas unidades de cinta. Puede utilizar un adaptador de canal a canal para enlazar los dos módulos UCP y cada módulo UCP tiene algunos canales privados conectados a algunos dispositivos de E/S privados y secundarios.



7.4 Sistema biprocesador IBM estrechamente acoplado con una única copia de sistema operativo residente en la memoria compartida [29]

[29] HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, p.748



7.5 Sistema biprocesador IBM estrechamente acoplado, ligeramente acoplado con un monoprocesador IBM [30.]

7.2 EL SISTEMA IBM 3033

El complejo multiprocesador 3033 cuenta con:

- Dos procesadores 3033
- Un modelo M
- Dos consolas 3036
- Una unidad de comunicación multiprocesador (MCU) 3038.

[30.] HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, p.749

El complejo multiprocesador asociado 3033 cuenta con:

- Un procesador 3033
- Un modelo A
- Un procesador asociado 3042
- Dos consolas 3036
- Una unidad de comunicación multiprocesador (MCU) 3038.

La figura 7.6 muestra una relación entre la MCU y las funciones del procesador.

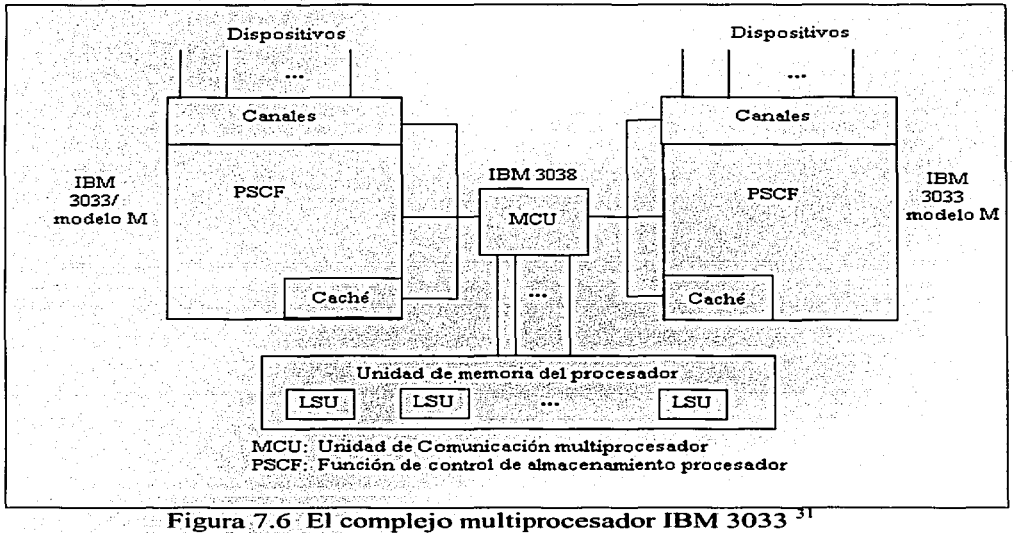


Figura 7.6 El complejo multiprocesador IBM 3033 [31]

[31] HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, p.749

La MCU con multiprocesador asociado ofrece:

- Prefijación.
- Comunicación interprocesador.
- Comunicación de actualización caché (memoria intermediaria de alta velocidad) y memoria.
- Compartición de memoria del procesador.
- Control de la configuración de particiones.
- Facilidades de sincronización.
- Comunicación de cambios de las claves de protección de memoria.

La MCU capacita a los dos procesadores una configuración MP para que puedan acceder a toda la memoria del procesador durante la retención de la capacidad de solapamiento en operaciones de memoria debido al entrelazado de cuatro vías. Significa que los dos procesadores pueden tener operaciones de memoria concurrentes con un grado variable de solapamiento dependiendo de la secuencia particular de accesos LSU. El control de la configuración y configurado en un sistema MP presenta varias opciones de configuración de memoria que pueden distribuidas a la memoria independientemente a cada procesador para el modo multiprocesador .

El complejo multiprocesador 3033 MP obtiene:

- Alto rendimiento, mediante una comunicación interprocesador más elevada

- Mejor caché
- Protección de memoria
- Mayor rapidez, en el acceso a la memoria del procesador compartida que el 370/168 MP.

7.3 EL SISTEMA 3081 IBM

El sistema 3081 presenta una organización asimétrica de dos procesadores centrales, cada uno con un tiempo de ciclo maquina de 26 nseg, y ejecuta instrucciones IBM 370 aproximadamente dos veces a la velocidad de IBM 3033.

Los objetivos del sistema 3081 son :

- Mejorar el índice precio/rendimieto sobre el sistema 370 y las series 303sx.
- Elevar la compatibilidad con aquellas líneas de productos.

Fue diseñado para mejorar la fiabilidad, disponibilidad y la facilidad de mantenimiento a través de la nueva tecnología y esquemas de particionamiento y encapsulamiento.

La principal consecuencia del encapsulado es el desarrollo de una unidad reemplazable de campo denominada módulo de conducción termal (TCM, Termal Conduction Module).
La figura 7.7 muestra la organización de la unidad procesadora 3081que presenta cinco subsistemas:

- Dos procesadores centrales
- El controlador del sistema
- La memoria principal
- El controlador de datos externo (EXDC) .

Al 3081 se le conoce como procesador didáctico, ya que esta configurado con dos procesadores idénticos que comparten un controlador de sistemas y EXDC dentro de una estructura. Actúa como un multiprocesador estrechamente acoplado que no puede ser desacoplado para actuar como dos monoprocesadores independientes como el IBM 3033.

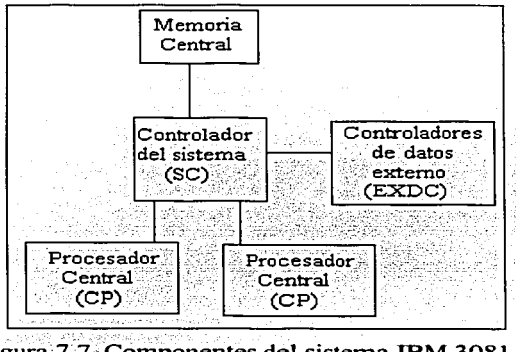


Figura 7.7 Componentes del sistema IBM 3081 [32]

[32] HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, p.751

La configuración es simétrica, puesto que cada procesador tiene la misma prioridad y características operacionales con respecto a la memoria central y canales.

Cada procesador tiene acceso a canales y a memoria central a través del controlador. Los procesadores comparten la memoria principal, que puede ser de 16, 24 o 32 Mb. Un área del sistema es el segmento de memoria principal, pero el área del sistema no es accesible a los programas de usuario. La memoria principal está organizada como un paquete de tarjetas y es entrelazada de doble vía. Cada placa, contiene 4M bytes de memoria principal, se denomina módulo de almacenamiento básico. Este módulo está configurado de manera que un bloque de datos (128 bytes) pueda ser accedido con una única operación para transferir un bloque entre el procesador y memoria.

El esquema de entrelazado utiliza:

- Doble palabra, es la unidad básica de operaciones de memoria.
- Una dirección de 2 k bytes a través de los módulos de 4 M bytes.

El segmento de 2 K bytes, se eligió para minimizar la reconfiguración de memoria, pudiendo de esta manera ser accedido independientemente.

El controlador del sistema proporciona los caminos y controla las comunicaciones entre la memoria principal y otros subsistemas. El ancho del bus de datos de todas las unidades que se conectan al controlador es de 8 bytes, con transferencia de datos de 8 bytes por ciclo máquina. El bus es bidireccional. El controlador también contiene las claves de protección de

memoria y reloj de la hora del día, maneja una cola de ocho posiciones que contiene peticiones de almacenamiento.

El 3081 soporta hasta 24 canales, que pueden ser de tipo multiplexor por byte o multiplexor por bloque. Los canales son controlados por el EXDC; el EXDC consta de dos tipos de elementos controlados por microcódigos. Uno de los elementos maneja el control de las instrucciones de E/S e interrupciones. El otro maneja el secuenciamiento del control de los datos y proporciona memoria intermedia para cada grupo de ocho canales.

Cada procesador consta de cinco elementos funcionales, como se muestra en la figura 7.8, y encapsulado dentro de una placa de nueve TCM. El procesador es segmentado como el IBM sistema 370/168. Sin embargo, tiene una capacidad de prebúsqueda de la instrucción.

Cada procesador tiene tres elementos de ejecución separados:

1. Elemento de control de la memoria intermedia
2. Elemento de almacenamiento de control.
3. Elementos de ejecución son los elementos de instrucción, elemento de campo variable y elemento de ejecución.

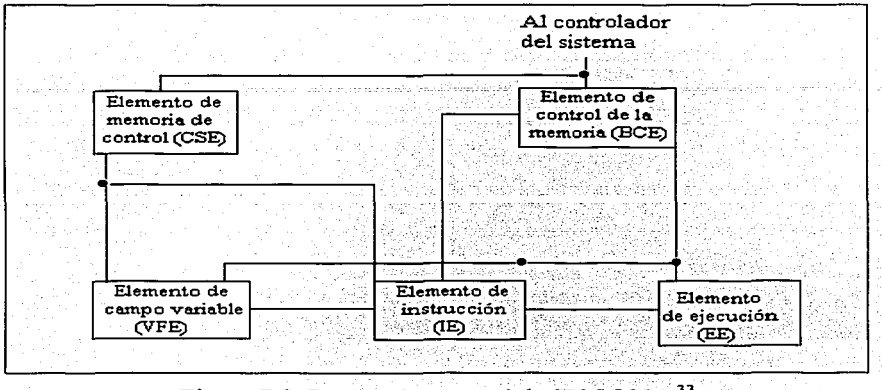


Figura 7.8 Procesador central de IBM 3081 [33]

El elemento de instrucción controla el secuenciamiento de las instrucciones de un procesador iniciando peticiones de instrucciones para tratar de mantener una memoria intermedia de instrucciones.

Ejecuta las funciones de decodificación de la instrucción y generación de la dirección del operando e inicializando todas las peticiones de operandos y realiza todas las operaciones aritméticas y lógicas.

El elemento de campo variable opera bajo control de microcódigo horizontal, ejecutando todas las instrucciones de longitud variable de memoria a memoria. Existe un sumador decimal dentro y las regiones de entrada y

[33] HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, p.751

salida son asociadas donde ejecuta instrucciones de conversión de tipo (binario a decimal) e instrucciones en punto flotante.

El elemento de control de la memoria intermedia contiene la caché de 32K o 64K bytes, La medida del bloque es de 128 bytes y la caché es asociativa por un conjunto de cuatro vías con un algoritmo de reemplazo de un tipo menos utilizado recientemente. La política de postescritura que utiliza el 3081 no es igual a la política de escritura utilizada en el 3033.

El estudio de simulación muestra que la política de postescritura era mejor sobre el 3081 a partir del tiempo de acceso a la memoria que excede es aproximadamente de diez ciclos máquina. La política de postecritura facilitó el algoritmo de reprocesamiento a partir de puntos de control (checkpoint retry) para el procesador. El de reprocesamiento es un dispositivo de hardware que se puede utilizar para establecer un punto de control en alguna instrucción *N*. En esta instrucción se salvan en una pila, los contenidos de los registros del procesador. Si ocurre una condición de error, los registros del procesador se pueden restaurar al estado original de su último punto de control y el proceso puede empezar de nuevo.

Sin embargo, cada procesador tiene un caché, a diferencia del esquema intel 3081 consiste en que un procesador pide la búsqueda de un bloque que es mantenido en exclusiva y modificado en caché, la copia del bloque se actualiza en memoria. Sin embargo, la copia en la caché es invalida en lugar de cambiar de su estado a sólo lectura. Enseguida, el procesador local busca el bloque desde memoria.

Un algoritmo predice que el uso de un bloque de datos fue desarrollado para reducir el recargo de los cambios de estado y de invalidación. Se puede configurar el sistema con cuatro procesadores (IBM 3084) utilizando un conjunto de dos 3081 en el modo MP o AP.

Conclusión

La hipótesis se probo estableciendo lo siguiente, los sistemas multiprocesadores con la arquitectura que presentan pueden ser la base de un mayor número de plataformas que en un sistema normal. Logrando el objetivo de determinar el desarrollo que han tenido los Sistemas Multiprocesadores, conocer y comprender la arquitectura así como las plataformas en las que se pueden desempeñar.

BIBLIOGRAFÍA

DONOVAN, Madnick, ***Operating systems***
7 th printing, Tokyo-Japon , Tosho Printing Co. LTD
1992, 640 pp.

HAWANG, Kai / A.BRIGGS, Fayé, ***Arquitecturas de computadoras y procesamiento paralelo,*** México, Editorial McGraw-Hill, 1988, 914 pp.

J.VERXELLO/REUTTER III, John, Robert
Procesamiento de datos conceptos y sistemas, México,
Editorial McGraw-Hill, 1986, 579 pp.

L.BECK, Leand, ***Software de sistemas introducción a la programación de sistemas,*** México, Editorial, Addison-Wesley Iberoamericana, 1985, 448 pp.

MICROSOFT ***Manual del usuario y referencia para el sistema operativo,***
MS-DOS Versión 5.0, México , Microsoft Corporation,
1991, 764 pp.

MILENKOVIC, Milan ***,Sistemas operativos conceptos y diseño,***
2ª ed., España, Editorial McGraw-Hill, 1994, 827 pp.

M.MORRIS, Mano, ***Arquitectura de computadoras***
3ª ed., México, Editorial PHH, Prentice Hall Hispanoamericana
1994, 563 pp.

SQUIRE, Enid, ***Introducción al diseño de sistemas,***
México, Editorial Alfaomega, 1990, 345 pp.

TANENBAUM S., Andrew , ***Sistemas operativos:diseño e implementación,***
México, Editorial PHH, Prentice Hall Hispanoamericana, 1987, 741 pp.

TANENBAUM S., Andrew, ***Organización de computadoras un enfoque estructurado,*** 3ª ed., México, Editorial PHH, Prentice Hall Hispanoamericana, 1992, 658 pp.

TANENBAUM S., Andrew, ***Organización de computadoras un enfoque de computadoras***, México, Editorial PHH, Prentice Hall Hispanoamericana 2000, 688 pp.

OTRAS FUENTES

http://www.aguila.itamx/bdd/unit.7htm.