8 Traducción de Direcciones

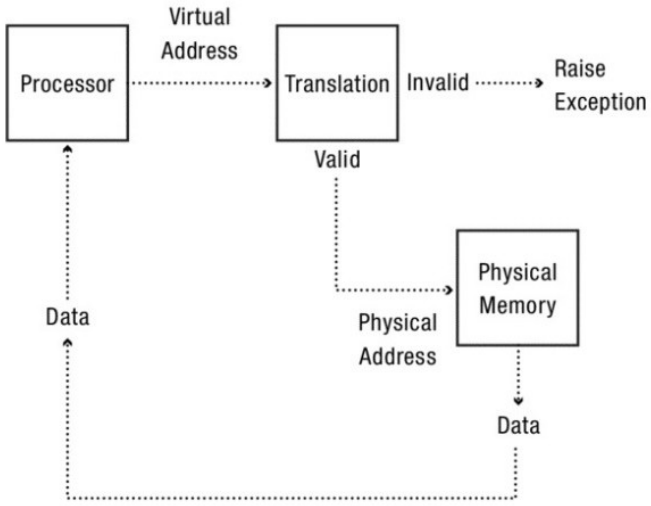
¿Podemos crear un entorno de realidad virtual para los programas de ordenador? Ya hemos visto un ejemplo de esto con la interfaz de E/S UNIX, cuando el programa no necesita saber, ya veces no se puede decir, si sus entradas y salidas son archivos, dispositivos, u otros procesos.
En los próximos tres capítulos, tomamos esta idea un paso de gigante. Un increíble número de opciones avanzadas del sistema están habilitados al poner el sistema operativo en el control de la traducción de direcciones, la conversión de la dirección de memoria del programa piensa que hace referencia a la ubicación física de la celda de memoria. Desde la perspectiva del programador, traducción de direcciones ocurre de forma transparente - el programa se comporta correctamente a pesar de que su memoria se almacena en algún lugar completamente diferente desde donde se cree que se almacena.
Probablemente se les enseñó en alguna clase de programación temprana de que una dirección de memoria es sólo una dirección. Un puntero en una lista enlazada que contiene la dirección de memoria real de lo que está apuntando. Una instrucción de salto contiene la dirección de memoria actual de la siguiente instrucción a ejecutar. Esta es una ficción útil. A menudo es mejor no pensar en cómo cada referencia a la memoria se convierte en los datos o instrucciones a las que se hace referencia. En la práctica, hay una gran cantidad de actividades que ocurre bajo las mantas.
La traducción de direcciones es un concepto simple, pero resulta ser increíblemente poderoso. ¿Qué puede hacer un sistema operativo con la traducción de direcciones? Esta es solo una lista parcial:

- Aislamiento de procesos.
- La comunicación entre procesos.
- Segmentos de código compartido.
- El inicio del programa.
- Asignación eficiente de memoria dinámica.
- Gestión de la caché.
- Distribuir la memoria compartida

Para una mayor eficiencia en tiempo de ejecución, la mayoría de los sistemas tienen hardware especializado para hacer la traducción de direcciones; este hardware es administrado por el kernel del sistema operativo. En algunos sistemas, sin embargo, la traducción es proporcionada por un gran compilador, enlazador o intérprete de código. En otros sistemas, la aplicación realiza la traducción del puntero como una manera de gestionar el estado de las estructuras de datos propias. Aún en otros sistemas, se usa un modelo híbrido donde las direcciones se traducen tanto en software y hardware. La elección es a menudo un comercio de la ingeniería entre rendimiento, flexibilidad y costo. Sin embargo, la funcionalidad proporcionada a menudo es la misma, independientemente del mecanismo utilizado para implementar la traducción. En este capítulo, vamos a cubrir una serie de mecanismos de hardware y software.

8.1 El Concepto de Traducción de Direcciones

Considerada como una caja negra, la traducción de direcciones es una función simple, como en la figura. El traductor toma cada instrucción y la memoria de datos de referencia generada por un proceso, comprueba si la dirección es legal, y la convierte en una dirección de memoria física que se puede utilizar para traer o almacenar instrucciones o datos. Los datos en sí mismos se devuelve como están; no se transforman de ninguna manera. La traducción se suele implementar en hardware, y el núcleo del sistema operativo configura el hardware para lograr sus objetivos.



La tarea de este capítulo es la de rellenar los detalles acerca de cómo funciona ese cuadro negro. Si nos preguntamos ahora cómo es posible ponerlo en práctica. Si usted dijo que podíamos usar una matriz, un árbol, o una tabla hash, estaría en lo correcto - todos estos enfoques han sido tomados por los sistemas reales.
Teniendo en cuenta que son posibles diferentes implementaciones, ¿cómo debemos evaluar las alternativas? Éstos son algunos de los objetivos que podríamos querer obtener de un cuadro de traducción; el diseño dependerá de cómo equilibramos entre estos diversos objetivos.

- Protección de la Memoria. Necesitamos la capacidad de limitar el acceso de un proceso para ciertas regiones de la memoria, por ejemplo, para evitar que el acceso a la memoria no es propiedad del proceso.
- Compartir la Memoria. Queremos permitir que varios procesos compartan regiones seleccionadas de la memoria. Estas regiones compartidas pueden ser grandes (por ejemplo, si estamos compartiendo segmento de código de un programa entre varios procesadores que ejecutan el mismo programa) o relativamente pequeño (por ejemplo, si estamos compartiendo una biblioteca común, un archivo o una estructura de datos compartida).
- La Colocación Flexible de la Memoria. Queremos permitir que el sistema operativo dé la flexibilidad para colocar un proceso (y cada parte de un proceso) en cualquier lugar en la memoria física; esto nos permitirá hacer los grupos de memoria física de manera más eficiente.
- Direcciones Dispersas. Muchos programas tienen múltiples regiones de memoria dinámica que puede cambiar de tamaño en el transcurso de la ejecución del programa: el montón de objetos de datos, una pila para cada hilo, y la memoria de archivos mapeados.
- Eficiencia de las Operaciones de búsqueda en tiempo de ejecución. traducción de direcciones de hardware en cada extracción de instrucción y cada carga de datos y almacenar. No sería práctico si una búsqueda tomó, en promedio, mucho más tiempo para ejecutar que la instrucción en sí.
- Tablas de Traducción Compactas. También queremos que el espacio arriba de la traducción sea mínima; cualquier estructura de datos que necesitamos deben ser pequeñas en comparación con la cantidad de memoria física que se está gestionando.
- Portabilidad. Diferentes arquitecturas de hardware hacen diferentes opciones en cuanto a cómo implementan la traducción.

Vamos a terminar con un mecanismo de traducción de direcciones bastante complejo, y por lo tanto nuestra discusión a empezar con los mecanismos más simples posibles y añadir funcionalidad sólo cuando sea necesario. Será de gran ayuda durante el debate para que usted pueda tener en cuenta los dos puntos de vista de la memoria: el proceso ve su propia memoria, utilizando sus propias direcciones. Vamos a llamar a estas direcciones virtuales, porque no se corresponden necesariamente con ninguna realidad física. Por el contrario, para

el sistema de memoria, sólo hay direcciones físicas - localizaciones reales en la memoria. Desde la perspectiva del sistema de memoria, se le da direcciones físicas y lo hace Búsquedas de y almacena valores. El mecanismo de traducción convierte entre los dos puntos de vista: desde una dirección virtual a una dirección de memoria física.

8.2 Traducción Flexible de Direcciones

La traducción de direcciones de hardware consta de dos pasos. En primer lugar, ponemos el tema de la eficiencia de búsqueda a un lado, y en su lugar consideramos los otros objetivos mencionados anteriormente: asignación flexible de la memoria, la eficiencia del espacio, de protección y de intercambio, y así sucesivamente. Una vez que tenemos las características que queremos, a continuación, añadimos los mecanismos para recuperar la eficiencia de las operaciones de búsqueda.
Se ha ilustrado (capítulo 2) el concepto de protección de la memoria del hardware usando el hardware más simple: base y límites. El cuadro de traducción consiste en dos registros adicionales por proceso. El registro de la base especifica el inicio de la región del proceso de la memoria física; el registro consolidado especifica el alcance de esa región. Si se añade el registro de base a todas las direcciones generadas por el programa, entonces ya no es necesario un cargador de reubicación - las direcciones virtuales del programa empiezan desde 0 y aumentan de a saltos, y para las direcciones físicas se parte de la base y vamos a la base + cota. Dado que la memoria física puede contener varios procesos, el núcleo restablece el contenido de la base y delimita registros de cada cambio de contexto procesando los valores apropiados para ese proceso.
La traducción usando base y límites traducción es a la vez simple y rápido, pero carece de muchas de las características necesarias para apoyar los programas modernos. Esta metodología de traducción sólo es compatible con la protección de grano grueso a nivel de todo el proceso; no es posible evitar que un programa sobrescriba su propio código, por ejemplo. También es difícil compartir regiones de memoria entre dos procesos. Puesto que la memoria para un proceso tiene que ser contigua, apoyar a las regiones de memoria dinámica, como por montones, pilas de subprocesos, o archivos de memoria asignada, se vuelve difícil o imposible.

8.2.1 Memoria Segmentada o Segmentación de Memoria

Muchas de las limitaciones del uso de base y límites para la traducción se pueden remediar con un pequeño cambio: en lugar de mantener un solo par de registros base y límites por proceso, el hardware puede soportar una matriz de pares de estos registros, para cada proceso. Esto se denomina segmentación. Cada entrada en la matriz controla una porción o segmento, del espacio de direcciones virtuales. La memoria física para cada segmento se almacena de forma contigua, pero diferentes segmentos se pueden almacenar en diferentes lugares. **Los bits de orden superior de las direcciones virtuales se utilizan para índice en la matriz**; **el resto de la dirección es añadido a la base y se compara con el almacenado unido a dicho índice**. Además, el sistema operativo puede asignar diferentes segmentos de permisos diferentes, por ejemplo, para permitir la ejecución de sólo el acceso al código y de lectura-escritura de acceso a los datos. Aunque cuatro segmentos se muestran en la figura, en general, el número de segmentos se determina por el número de bits para el número de segmento que están a un lado en la dirección virtual.
La memoria segmentada tiene lagunas; la memoria del programa ya no es una sola región contigua, sino que es un conjunto de regiones. Cada segmento diferente comienza en un nuevo límite de segmento. Por ejemplo, el código y los datos no son inmediatamente adyacentes entre sí, ya sea en el espacio de dirección virtual o física.
¿Qué ocurre si un programa se ramifica en o intenta cargar datos de una de estas brechas? El hardware generará una excepción, atrapando en el núcleo del sistema operativo. En sistemas UNIX, esto todavía se llama un **fallo de segmentación**, es decir, una referencia fuera de un

segmento legal de la memoria. ¿Cómo mantener un programa de vagar en una de estas lagunas? los programas correctos no generarán referencias externas de memoria válida. Dicho de otra manera, intentar ejecutar código o datos de lectura que no existe es probablemente una indicación de que el programa tiene un error en ella.
Aunque simple de implementar y administrar, memoria segmentada es a la vez muy poderosa y ampliamente utilizado. Por ejemplo, se segmenta la arquitectura x86 (con algunas mejoras que describiremos más adelante). Con segmentos, el sistema operativo puede permitir a los procesos comparten algunas regiones de la memoria, manteniendo otras regiones protegidas. Por ejemplo, dos procesos pueden compartir un segmento de código mediante la creación de una entrada en sus tablas de segmentos para que apunte a la misma región de la memoria física - para utilizar la misma base y límites. Los procesos pueden compartir el mismo código mientras se trabaja fuera de datos diferentes, mediante la creación de la tabla de segmentos para que apunte a las diferentes regiones de memoria física para el segmento de datos.

También podemos utilizar segmentos para la comunicación entre procesos, de los procesos se dan los permisos para leer y escribir en el mismo segmento.
Como último ejemplo del poder de los segmentos, permiten la gestión eficiente de la memoria asignada dinámicamente. Cuando un sistema operativo vuelve a utilizar la memoria o espacio en disco que había sido utilizado anteriormente, se debe poner a cero primero el contenido de la memoria o disco. De lo contrario, los datos privados de una aplicación sin querer podrían filtrarse a otra, potencialmente malicioso, aplicación. Por ejemplo, podría introducir una contraseña en una página web, por ejemplo, para un banco, y luego salir del navegador. Sin embargo, si la memoria física subyacente utilizado por el navegador se vuelve a asignar a un nuevo proceso, a continuación, la contraseña puede ser filtrada a un sitio web malicioso.
La desventaja principal de la segmentación es la sobrecarga de gestión de un gran número de tamaño variable y de crecimiento dinámico segmentos de memoria. Con el tiempo, ya que los procesos son creados y el acabado, la memoria física se divide en regiones que están en uso y regiones que no son, es decir, disponible para ser asignado a un nuevo proceso. Estas regiones libres serán de diferentes tamaños. Cuando se crea un nuevo segmento, tendremos que encontrar un lugar libre para ello. ¿Hay que ponerlo en la región abierta más pequeña donde se forma? ¿La región abierta más grande?
Sin embargo, optamos por colocar nuevos segmentos, como más memoria asignada, el sistema operativo puede llegar a un punto donde hay suficiente espacio libre para un nuevo segmento, pero el espacio libre no es contiguo. Esto se llama **fragmentación externa**. El sistema operativo es libre de memoria compacta para hacer espacio sin necesidad de aplicaciones que afectan, porque las direcciones virtuales son sin cambios cuando reubicar una serie de sesiones en la memoria física. Aun así, la compactación puede ser costoso en términos de sobrecarga del procesador: una configuración de servidor típica tomaría aproximadamente un segundo para compactar la memoria

8.2.1 Memoria Paginada o Paginación de Memoria

Una alternativa a la memoria segmentada es la memoria paginada. Con la paginación, la memoria se asigna en bloques de tamaño fijo llamados **frames o marcos** de página. La traducción de direcciones es similar a la forma en que se trabaja con la segmentación. En lugar de una tabla de segmentos cuyas entradas contienen punteros a los segmentos de tamaño variable, hay una **tabla de páginas** para cada proceso cuyas entradas contienen punteros a los marcos de página. Debido a que los marcos de página fijan el tamaño en una potencia de dos, las entradas de la tabla de páginas sólo tienen que proporcionar los bits superiores de la dirección del marco de página, por lo que son más compactos. No hay necesidad de un "límite" en el desplazamiento; toda la página en la memoria física se asigna como una unidad.
Algo raro sobre la paginación es que mientras un programa piensa en su memoria en forma lineal, de hecho, su memoria puede estar, y normalmente esta, dispersa por toda la memoria física en una especie de mosaico abstracto. El procesador ejecutará una instrucción tras otra utilizando direcciones virtuales; sus direcciones virtuales son todavía lineales. Sin embargo, la instrucción situada en el extremo de una página se encuentra en una región completamente

diferente de la memoria física de la siguiente instrucción en el inicio de la siguiente página. Las estructuras de datos parecen estar contiguas mediante direcciones virtuales, pero una gran matriz puede ser distribuida en varios marcos de página físicos.
La paginación es la principal limitación de la segmentación: la asignación de espacio libre es muy sencilla. El sistema operativo puede representar memoria física como un mapa de bits, con cada bit representa un marco de página física que ya sea libre o en uso. Encontrar un marco libre es sólo una cuestión de encontrar un poco vacío.
Compartir la memoria entre procesos es también conveniente: tenemos que establecer la entrada en la tabla de páginas para cada proceso de intercambio de una página para que apunte al mismo marco de página física. Para una gran región compartida que abarca varios marcos de página, como una biblioteca compartida, puede ser necesaria la creación de un número de entradas en la tabla de páginas. Puesto que necesitamos saber cuándo liberar memoria cuando un proceso termina, la memoria compartida requiere un poco de contabilidad adicional para mantener un seguimiento de si la página compartida todavía está en uso. La estructura de datos para esto se llama un mapa de núcleo; que registra información sobre cada marco de página físico, como el que las entradas de la tabla de páginas apuntan a la misma.
Las tablas de páginas permiten añadir otras características. Por ejemplo, podemos iniciar un programa que se ejecuta antes de que todo su código y los datos se cargan en la memoria. Inicialmente, el sistema operativo marca todas las entradas de la tabla para un nuevo proceso como no válido; como las páginas son traídas del disco, marca aquellas páginas que son de solo lectura (para páginas de códigos) o de lectura y escritura (para páginas de datos). Sin embargo, una vez que las primeras páginas están en memoria, el sistema operativo puede iniciar la ejecución del programa en modo de usuario, mientras que el kernel sigue transfiriendo el resto del código del programa en el fondo. A medida que el programa se pone en marcha, si sucede un saltar a un lugar que no ha sido cargado todavía, el hardware provocará una excepción, y el núcleo puede detener el programa hasta que la página está disponible. Además, el compilador puede reorganizar el programa ejecutable para un inicio más eficiente, por coalescencia de las páginas de inicialización en algunas páginas en el inicio del programa, así inicialización superposición y cargar el programa desde el disco.
Como otro ejemplo, un **punto de interrupción de datos** es solicitar detener la ejecución de un programa cuando se hace referencia o modifica una posición de memoria particular. Es muy útil durante la depuración para saber cuándo una estructura de datos ha cambiado, sobre todo cuando se estén rastreando errores de puntero. los puntos de interrupción de datos a veces se implementan con el apoyo de hardware especial, pero también pueden ser ejecutadas con tablas de páginas. Para ello, la entrada de tabla de la página que contiene la ubicación se marca como de sólo lectura. Esto hace que el proceso de trampa para el sistema operativo en cada cambio de la página; el sistema operativo puede entonces revisar si la instrucción que está causando la excepción afectó a la ubicación específica o no.
Una desventaja de la paginación es que mientras que la gestión de memoria física se hace más simple, la gestión del espacio de direcciones virtuales se hace más difícil. Los compiladores normalmente esperan que la pila de ejecución sea contigua (en las direcciones virtuales) y de tamaño arbitrario; cada nueva llamada a procedimiento asume que la memoria de la pila está disponible. Del mismo modo, la biblioteca de tiempo de ejecución para la asignación dinámica de memoria por lo general espera una pila contigua. En un único subproceso, podemos colocar la pila y el montón en los extremos opuestos del espacio de direcciones virtuales, y tienden a crecer hacia la otra. Sin embargo, con múltiples hilos por proceso, necesitamos múltiples pilas de subprocesos, cada una con espacio para crecer.
Esto empeora con los espacios de direcciones virtuales de 64 bits. El tamaño de la tabla de páginas es proporcional al tamaño del espacio de direcciones virtuales, no al tamaño de la memoria física. Cuanto más escaso el espacio de direcciones virtuales, más se necesita la sobrecarga de la tabla. La mayor parte de las entradas no será válida, en representación de las partes del espacio de direcciones virtuales que no están en uso, pero la memoria física aún es necesaria para todas esas entradas de la tabla.

Podemos reducir el espacio ocupado por la tabla de páginas por elegir un marco de página más grande. ¿Qué tamaño debería tener un marco de página? Un marco de página más grande puede desperdiciar espacio si un proceso no utiliza toda la memoria dentro del marco. Esto se llama **fragmentación interna**. Trozos de tamaño fijo son más fáciles de asignar, pero el espacio de desperdicia si no se utiliza todo el trozo. Desafortunadamente, esto significa que, con paginación, ya sean páginas muy grandes (perder el espacio debido a la fragmentación interna) o la tabla de páginas es muy grande (desperdiciar espacio), o ambos. ¡Por ejemplo, con 16 páginas KB y un espacio virtual de direcciones de 64 bits, puede ser que necesite 250 entradas de tabla de página!

8.2.3 Traducción Multinivel

Si se va a diseñar un sistema eficiente para hacer una búsqueda en un espacio de claves, probablemente no volvería a escoger una matriz simple. Un árbol o una tabla hash son más apropiados, y, de hecho, los sistemas modernos usan ambos. Nos centramos en este apartado en los árboles; discutimos tablas hash después.
Muchos sistemas utilizan traducción de direcciones basado en árboles, aunque los detalles varían de un sistema a otro, y la terminología puede ser un poco confusa. A pesar de las diferencias, los sistemas que vamos a describir tienen propiedades similares. Apoyan la protección de grano fino y grueso de memoria y la memoria compartiendo, la colocación flexible de la memoria, la asignación de memoria eficiente y eficaz para las operaciones de búsqueda de espacios de direcciones escasos.
Casi todos los sistemas de traducción de direcciones de varios niveles utilizan paginación como el nivel más bajo del árbol. Las principales diferencias entre los sistemas están en la forma en que llegan a la mesa de página en la hoja del árbol - si el uso de los segmentos más paginación, o múltiples niveles de paginación, o segmentos, además de múltiples niveles de paginación.

Hay varias razones para esto:

- La Asignación Eficiente de la Memoria. Con la asignación de memoria física en marcos de página de tamaño fijo, la gestión de espacio libre puede utilizar un mapa de bits simple.
- Transferencias de Disco Eficiente. discos de hardware se dividen en regiones de tamaño fijo llamados sectores; sectores del disco deben ser leídos o escritos en su totalidad. Al hacer que el tamaño de la página un múltiplo del sector de disco, simplificamos las transferencias desde y hacia la memoria, para los programas de carga en la memoria, la lectura y escritura de archivos, y en el uso del disco para simular una memoria más grande que está físicamente presente en la máquina.
- Operaciones de Búsqueda Eficiente.
- Búsqueda Inversa Eficiente. El uso de marcos de página de tamaño fijo también hace que sea fácil de poner en práctica el mapa principal, para pasar de un marco de página física para el conjunto de direcciones virtuales que comparten el mismo marco. Esto será crucial para la aplicación de la ilusión de una memoria virtual infinito.
- Protección de Página-Granularidad y Compartir. Por lo general, cada entrada de la tabla en cada nivel del árbol tendrá sus propios permisos de acceso, lo que permite el intercambio tanto de grano grueso y de grano fino, hasta el nivel del marco de página individual.

Vamos a empezar con un sistema de sólo dos niveles de un árbol. Con la segmentación paginada, la memoria es segmentada, pero en lugar de que cada entrada de la tabla de segmentos apunte directamente a una región contigua de memoria física, cada entrada de la tabla de segmentos apunta a una tabla de páginas, que a su vez apunta a la copia de

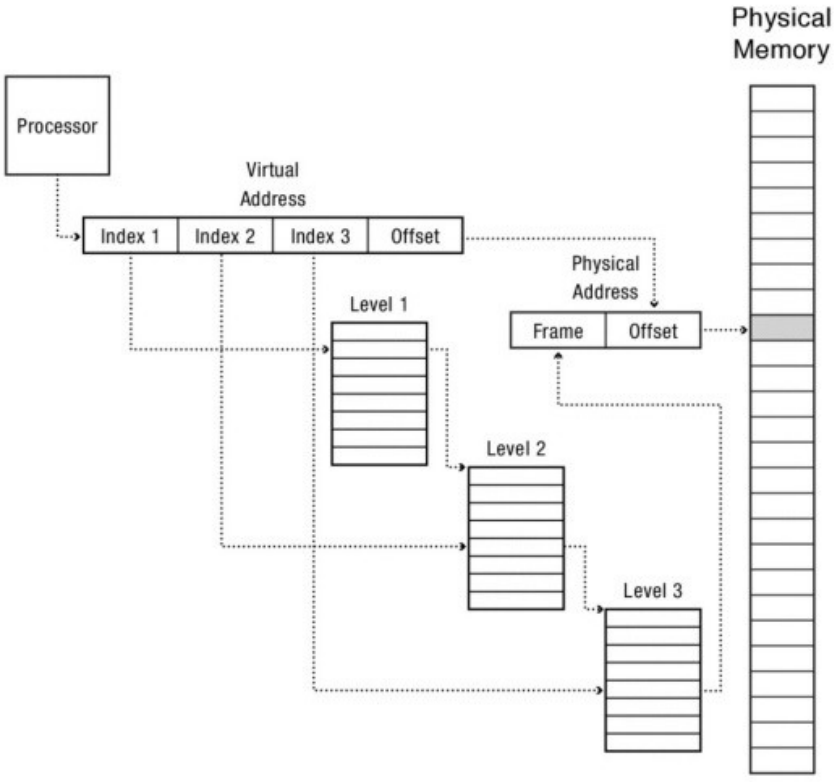
memoria de ese segmento. La entrada de la tabla segmento "unido" describe la longitud de la tabla de página, es decir, la longitud del segmento en las páginas. Debido a la paginación que se utiliza en el nivel más bajo, todas las longitudes de segmento son un múltiplo del tamaño de página.

A pesar de tablas de segmentos a veces se almacenan en registros especiales de hardware, las tablas de páginas para cada segmento son un poco más grande en su conjunto, por lo que normalmente se almacenan en la memoria física. Para mantener el asignador de memoria simple, el tamaño máximo de segmento se elige generalmente para permitir que la tabla de páginas para cada segmento sea un pequeño múltiplo del tamaño de página.

Por ejemplo, con las direcciones virtuales de 32 bits y páginas de 4 KB, podríamos dejar de lado los diez bits superiores para el número de segmento, los próximos diez bits para el número de páginas y los doce bits para el desplazamiento de página. En este caso, si cada entrada de la tabla de páginas es de cuatro bytes, la tabla de página para cada segmento sería encaja exactamente en un marco de página física.

Paginación Multinivel

Un enfoque casi equivalente a la segmentación paginada es utilizar múltiples niveles de tablas de páginas. Como se muestra en la figura, la tabla de páginas de nivel superior contiene entradas, cada una de las cuales apunta a una tabla de páginas de segundo nivel cuyas entradas son punteros a tablas de páginas. En los sistemas que utilizan varios niveles de tablas de páginas, cada nivel de la tabla de páginas está diseñado para encajar en un marco de página física. Sólo la tabla de páginas de nivel superior debe ser llenado; los niveles inferiores del árbol se asignan sólo si las partes del espacio de direcciones virtuales están en uso por un proceso particular. Los permisos de acceso pueden especificarse en cada nivel, y así compartir entre los procesos es posible en cada nivel.



Segmentación de Paginado Multinivel

Podemos combinar estos dos enfoques mediante el uso de una memoria segmentada, donde cada segmento es manejado por una tabla de páginas de varios niveles. Este es el enfoque adoptado por el x86, tanto por sus modos de direccionamiento de 32 bits y de 64 bits.

Describimos el caso de 32 bits en primer lugar. La terminología x 86 difiere ligeramente de lo que hemos utilizado aquí. El 86 tiene un proceso de per-Global Descriptor Table (GDT), lo que equivale a una tabla de segmentos. El GDT se almacena en la memoria; cada entrada son puntos a la tabla de páginas para ese segmento, junto con los permisos de acceso de longitud de segmento y segmento. Para iniciar un proceso, el sistema operativo crea el GDT e inicializa un registro, la tabla de descriptor Registro Mundial (GDTR), que contiene la dirección y la longitud del GDT.

Para codificar la eficiencia, el registro de segmento es a menudo implícito como parte de la instrucción. Por ejemplo, las instrucciones de pila x86 como de push() y pop() asumen el segmento de pila (el índice almacenado en el registro segmento de pila), instrucciones de ramificación asumen el segmento de código (el índice almacenado en el registro segmento de código), y así sucesivamente. Como una optimización, siempre que el 86 inicializa un código, pila, o un segmento de registro de datos también lee la entrada GDT (es decir, la tabla de la página de puntero y los permisos de acceso de nivel superior) en el procesador, por lo que el procesador puede ir directamente a la tabla de páginas en cada referencia.

Muchas instrucciones también tienen la opción de especificar el índice del segmento de forma explícita. Por ejemplo, el LJMP, o salto de longitud, la instrucción cambia el contador de programa a un nuevo número de segmento y el desplazamiento dentro de ese segmento.

Para el x86 de 32 bits, el espacio de direcciones virtuales dentro de un segmento tiene una tabla de páginas de dos niveles. Los primeros 10 bits de la dirección virtual índice de la tabla de páginas de nivel superior, llamado el directorio de páginas, el siguiente índice de 10 bits del segundo nivel de la tabla de páginas, y los últimos 12 bits son el desplazamiento dentro de una página. Cada entrada de la tabla de páginas toma cuatro bytes y el tamaño de la página es de 4 KB, por lo que la tabla de páginas de nivel superior y cada tabla de páginas de segundo nivel se ajusta en una sola página física. El número de tablas de páginas de segundo nivel necesarios depende la longitud del segmento; que no son necesarios para mapear regiones vacías del espacio de direcciones virtuales. Tanto el nivel superior y las entradas de la tabla de páginas de segundo nivel tienen permisos, por lo que es posible la protección de grano fino y el intercambio dentro de un segmento.