DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Tierra Pirata

Trabajo Práctico III - Programación de sistemas

7 de diciembre de 2017 — Organización del Computador II

Grupo: Ariane 5

Integrante	LU	Correo electrónico
Greco, Luis	150/15	luifergreco@gmail.com
Hertzulis, Nicolás	811/15	nicohertzulis@gmail.com
Ramos, Ricardo	841/11	riki_german@yahoo.com.ar

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Introducción

Un sistema operativo es el conjunto de programas de un dispositivo informático (laptop, celular, etc.) que maneja los recursos de hardware y provee servicios a los programadores de aplicaciones mediante interrupciones de software. Los sistemas operativos se ejecutan en modo privilegiado, aunque puede que una parte se ejecute en modo usuario.
El núcleo o kernel es la parte más fundamental del sistema operativo, que se encarga de hacer los primeros pasos para el arranque del sistema y de proveer las rutinas de atención a interrupciones de software, que utilizan los programadores de aplicaciones.
El kernel se define como la parte del sistema operativo que se ejecuta en modo privilegiado.
En este trabajo educativo nos proponemos hacer un sistema operativo muy básico para entender los fundamentos de la programación de sistemas operativos.
Para ello completamos diversos fragmentos de un esqueleto de kernel provisto por los docentes.

2. Desarrollo

2.1. Ejercicio 1

2.1.1. Segmentos de código, datos y de pantalla

El sistema operativo utiliza dos segmentos de código de nivel 0 y 3, dos segmentos de datos de nivel 0 y 3 y un segmento que describe el área de la pantalla de video. Declaramos los cinco segmentos en la Tabla de Descriptores Globales (GDT), cada uno con su correspondiente descriptor. La GDT es un arreglo en memoria cuyas entradas ocupan 64 bits y tienen una estructura particular (ver figura 1). Para ingresar los datos con comodidad utilizamos un struct de C. La primer entrada está ubicada en la posición con índice 8. Todos los segmentos excepto el de video direccionan los primeros 500 MB de la memoria, es decir que se superponen.

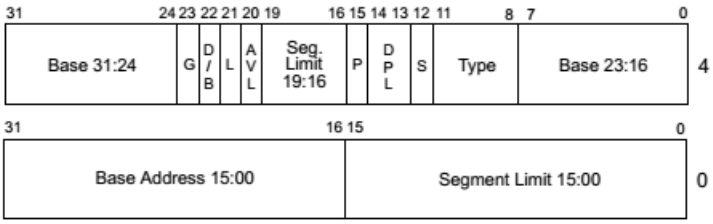

Figura 1: Descriptor de segmento.

- Base, límite y granularidad: La base es 0x0, el límite es 0x1F3FF y la granularidad es 1. La granularidad activada hace que tratemos a la memoria como bloques de 4K, por lo tanto el límite con granularidad corresponde a la cantidad de bloques de 4K menos uno. Para direccionar los primeros MB, necesitamos la base en cero. El límite es 0x1F3FF porque en decimal es 127999, ya que necesitamos 128000 bloques de 4KB para llegar a 500 MB. Todo esto es para los segmentos de código y datos. Para descriptor de segmento de pantalla debemos especificar como base `0xB8000` y para el límite necesitamos resolver (80 * 50 * 2) - 1, donde 80 es la cantidad de celdas, 1 celda == 2 bytes, en fila de pantalla y 50 es cantidad de celdas en columna. Es decir que el límite nos la cantidad de bytes que necesitamos para escribir en cualquier parte de la pantalla. Luego el valor del límite es `0x1F3F`. La granularidad está cero ya que tiene menor tamaño (no necesita bloques de 4KB para direccionar toda su memoria).

- Tipo: 0x0A en los segmentos de código (lectura, ejecución) y 0x02 en los segmentos de datos y de video (lectura, escritura).
- Nivel de privilegios: 0x0 en los segmentos de datos y código que utiliza en kernel, así como en el segmento de video (que también es de uso exclusivo del kernel). Estos son los segmentos de nivel cero mencionados anteriormente. 0x03 en los segmentos de código y datos que utilizan las tareas, o sea el usuario, que corresponden a los segmentos de nivel tres mencionados anteriorente.
- Sistema: 0x1 (que significa desactivado) pues no son segmentos de sistema sino de código y datos.
- Presente: 0x1.

2.1.2. Modo protegido y pila del kernel

Realizamos los siguientes pasos:

- Habilitar A20 para poder acceder a direccioner superiores a 2^{20} bits.
- Inicializar la GDT, cuyas entradas fueron ingresadas a partir de una dirección de memoria arbitraria, mediante la instrucción lgdt.
- Activar el bit menos significativo de CR0.
- Saltar a segmento:modoprotegido, donde segmento corresponde al índice del segmento de código de nivel 0 en la GDT corrido 3 ceros a la izquierda y modoprotegido corresponde a la dirección de memoria donde arranca el código que se ejecutará a continuación (ya en modo protegido).
- Establecer los registros selectores de segmento datos (ds, es, gs y ss) en el índice del segmento de datos de nivel cero en la GDT corrido 3 bits a la izquierda.
- Establecer el registro selector de segmento de video (fs) en el índice del segmento de video en la GDT corrido 3 bits a la izquierda.
- Establecer la base de la pila en 0x27000.

2.1.3. Limpieza e inicialización de la pantalla

Para limpiar la pantalla hicimos una función en ensamblador que utiliza el segmento de video y otra en C que utiliza el segmento de datos. La función en ensamblador recorre byte por byte y la función en C recorre celda por celda (2 bytes) y llama a la función screen_pintar en cada iteración. Ambas funciones dejan la pantalla en negro.
Para inicializar la pantalla llamamos a la función screen_inicializar que fue completada por nosotros, llamando a la función para limpiar la pantalla y luego a la función screen_pintar_rect para las distintas secciones.

2.2. Ejercicio 2

2.2.1. Rutinas de atención a las excepciones del procesador

Cada interrupción debe tener su entrada en la Tabla de Descriptores de Interrupción (IDT). Las primeras 32 posiciones de esta tabla (índices 0 a 31) corresponden a las excepciones del procesador.
Así como en la GDT, para las entradas de la IDT utilizamos un struct de C, cuyos campos y valores son los siguientes:

- Selector de segmento: 0x40 que corresponde al índice del segmento de código del kernel en la GDT corrido 3 bits a la izquierda (índice 8).
- Desplazamiento: La dirección donde comienza el código de la rutina de atención. Es una dirección distinta para cada rutina. Las rutinas están definidas con una macro en ensamblador cuyas etiquetas están declaradas como símbolos globales para poder referenciarlas luego desde C.
- Atributo: Es un campo que agrupa varios valores. En nuestro caso el valor es 0x8E00. El 8 es un valor hexadecimal que en binario es 1000. El 1 es el bit de presente, los dos bits siguientes son el DPL (nivel 00) y el bit menos significativo debe estar en cero para interrupciones. La E también es un valor hexadecimal que corresponde al *gatetype* y en este caso significa puerta de interrupción de 32 bits. Así, el valor de atributo es 0x8E y luego se agregan dos ceros a la derecha porque debe estar corrido siempre 8 bits (los ceros son fijos).

Las entradas de la IDT se cargan desde una dirección de memoria arbitraria. Una vez cargadas, inicializamos la IDT utilizando la instrucción lidt (load IDT).
Las rutinas solo muestran el número de excepción (que corresponde al índice de la IDT) y luego quedan en un ciclo infinito.

2.3. Ejercicio 3

2.3.1. Paginación

La paginación es un mecanismo que permite tratar a la memoria en bloques de tamaño fijo, que en este caso son de 4KB y asignar las direcciones físicas de cada página a un rango de mismo tamaño de memoria virtual.
Un rango de direcciones de tamaño de una página de memoria virtual puede estar asignado a distintas páginas de memoria física, cada asignación en un directorio de páginas distinto.
Un directorio de páginas tiene 1024 entradas que corresponden a 1024 tablas de páginas. Una tabla de páginas tiene 1024 entradas que corresponden a 1024 páginas.

2.3.2. Directorio y tablas de páginas del kernel

El kernel tiene su propio directorio de páginas ubicado en la dirección física 0x27000. Este se va a encargar de las páginas de la zona del kernel, que corresponde a las direcciones físicas 0x00000000 a 0x003FFFFF. Estas direcciones serán asignadas a otras direcciones de memoria virtual, que casualmente en este caso coinciden con las direcciones físicas, por eso decimos que es la asignación identidad, pues la función de asignación es la función identidad.
Para asignar la memoria debemos calcular cuántas páginas necesitamos. Para eso hacemos la siguiente cuenta: (0x003FFFFF + 0x1) / 0x1000 que es igual a 0x400. 0x1000 es el tamaño de una página (en decimal es 4096 pues una página es de 4KB). El resultado 0x400 en decimal es 1024. Por lo cual necesitamos 1024 páginas para completar la zona del kernel. Casualmente 1024 es el tamaño de una tabla de páginas, por lo cual el directorio de páginas del kernel tendrá todas las entradas vacías excepto la primera, que hará referencia a una tabla que estará completamente llena. La tabla de páginas estará en la dirección física 0x28000.
La función que se encarga de inicializar el directorio del kernel es la función mmu_inicializar_dir_kernel y realiza los siguientes pasos:

- Llenar las 1024 entradas del directorio en cero. En particular el bit de presente está en cero, que en nuestro sistema operativo significa que la memoria correspondiente a esa tabla no está asignada.
- Modificar la primera entrada con los siguientes valores: 1) Bit de presente activado. 2) Bit de lectura-escritura activado. 3) Base en 0x27 que es la dirección física de la tabla (0x27000) corrida 12 bits a la derecha.

- Para cada entrada de la tabla, poner una entrada con los siguientes valores: 1) Bit de presente activado. 2) Bit de lectura-escritura activado. 3) Base en el valor i donde i es un número entero entre 0 y 1023. La dirección física es en realidad (i << 12), pero como luego hay que volverla a correr 12 bits para ingresarla a la estructura, (i << 12) >> 12 es equivalente a i.

2.3.3. Activación de paginación

Para tener el mecanismo de paginación funcionando, realizamos los siguientes pasos:

- Realizar las asignaciones de memoria deseadas (como mínimo la zona del kernel) tal como explicamos en la sección anterior. En nuestro sistema operativo lo hacemos llamando a mmu_inicializar_dir_kernel.
- Poner la dirección física del directorio en el registro de control 3 (CR3).
- Activar el bit más significativo del registro de control 0 (CR0). Este es el paso que activa el mecanismo.

Durante la ejecución del sistema operativo se pueden modificar las asignaciones de memoria del directorio en uso o de otros directorios.

2.4. Ejercicio 4

2.4.1. Manejador de memoria

Tenemos un manejador de memoria al que le podemos pedir una página libre. El mecanismo es muy básico. Tenemos una variable inicializada en la posición física 0x100000. Cuando alguien llama a la función para pedir la dirección de una página libre, el manejador devuelve el valor de esta variable y luego la incrementa en 0x1000 que es el tamaño de una página, así en la próxima llamada a la función se devuelve la página que está inmediatamente más abajo, que se asume libre. Con este mecanismo, si borramos una página física de todos los directorios la misma no podrá volver a ser asignada nuevamente, porque el manejador no se enterará que está libre, ya que la variable de la próxima página libre siempre crece.

2.4.2. Asignación de páginas

La función mmu_mapear_pagina se encarga de asignar un rango de direcciones de memoria virtual a memoria física en un directorio determinado, siendo ese rango el correspondiente a una página. Las direcciones físicas asignadas serán, por lo tanto, el rango [A, A+0x999] donde A es la dirección de comienzo de la página (múltiplo de 0x1000) y 0x999 es 0x1000 - 0x1. No se puede asignar un bloque de memoria menor a una página. Si una página no es suficiente, se puede llamar a esta función tantas veces como sea necesario con distinta páginas. Notar que no es necesario que las páginas físicas sean continuas, pues solo importa que las direcciones virtuales sean contiguas. Esta función, entonces, recibe los siguientes parámetros:

- Dirección física de comienzo de la página (múltiplo de 0x1000).
- Dirección virtual con la que se desea hacer la asignación.
- Dirección virtual del directorio de páginas en donde se desea hacer la asignación.

Pasos que realiza la función para hacer la asignación:

- Tomar los 10 bits más significativos de la dirección virtual que corresponden al número de tabla en el directorio de tablas.
- Tomar los 10 bits siguientes de la dirección virtual que correspondenal número de página en la tabla de páginas.

- Los 12 bits menos significativos corresponen al desplazamiento dentro de la página, que se usan en el mecanismo de paginación para poder acceder a direcciones particulares dentro de una página. En este caso los desestimamos porque siempre se asigna la página en bloque (solo necesitamos la dirección de comienzo).
- Si la entrada en el directorio de páginas correspondiente a la tabla (accedemos con el número de tabla calculado anteriormente) tiene el bit de presente activado, obtenemos la dirección de la tabla de páginas.
- Si no está presente, establecemos como dirección de tabla una dirección de página libre que le pedimos al manejador de memoria y activamos el bit de presente. Las 1024 entradas de esta nueva tabla las inicializamos vacías (en cero).
- Ahora que tenemos la dirección de la tabla, accedemos a la entrada correspondiente a la página utilizando el número de página calculado al comienzo y pisamos los siguientes valores: activamos bit de presente y ponemos como dirección base la dirección física de la página que recibimos como parámetro.
- Finalmente llamamos a una función para invalidar la cache de traducción de direcciones.

2.4.3. Eliminación de páginas

La función mmu_desmapear_pagina se encarga de hacer que una página deje de pertenecer a un directorio determinado. Para hacer esto, simplemente separamos la dirección virtual en número de tabla y número de página como en la asignación, accedemos a la tabla desde el directorio y desactivamos el bit de presente de la página correspondiente desde la tabla. Adicionalmente, si detectamos que la tabla queda vacía (todas las entradas con el bit de presente desactivado), desactivamos el bit de presente de la tabla desde el directorio. Por último, llamamos a una función para invalidar la cache de traducción de direcciones.

2.4.4. Directorios de las tareas pirata

La función mmu_inicializar_dir_pirata se encarga de crear el directorio para una tarea pirata. Recibe como parámetros:

- Dirección física de la página donde se encuentra el código la tarea (se asume que entra en una sola página).
- Dirección física de destino del pirata (debería ser parte del mapa).
- ID del pirata.

Realiza los siguientes pasos:

- Pedir al manejador de memoria la dirección de una página libre que utilizaremos para el directorio.
- Asignar las páginas correspondientes a la zona del kernel en el nuevo directorio. Esto es para poder atender interrupciones.
- Asignar las páginas en posiciones matricialmente contiguas a la dirección de destino del código en el directorio actual pero usando una tabla de páginas compartida, lo cual hace que estas páginas sean visibles para todas las tareas automáticamente.
- Copiar el código del pirata de la dirección física de origen a la dirección física destino. Para hacer el copiado del código es necesario asignar una dirección virtual libre cualquiera que esté en la zona de kernel a la dirección física de destino del pirata y usar esta dirección temporal para copiar. Esto es necesario porque para hacer el copiado no usamos el nuevo directorio, sino el directorio actual, que puede pertenecer a otro pirata y si hacemos el copiado usando

directamente la dirección virtual 0x400000 estaríamos perdiendo la asignación de ese posible pirata.

- Luego de hacer el copiado, eliminar esta página temporal del directorio actual y asignar la dirección 0x400000 en el nuevo directorio.
- Devolver la dirección física del directorio de páginas de la tarea.