Trabajo práctico

Organización del Computador II

25 de mayo de 2009

Índice

1. Introducción

En este trabajo práctico se desarrollarán algunas funciones de un sistema operativo académico llamado JOS. El énfasis estará puesto en el manejo de memoria paginado, pues es una parte fundamental de los sistemas operativos modernos que utiliza en gran parte del soporte del procesador.

1.1. Objetivo

El objetivo de este trabajo es utilizar en un sistema operativo real los conceptos vistos en las clases teóricas relacionados con el soporte del procesador para el sistema operativo. A diferencia de las prácticas, en este caso se dispone de un sistema operativo completo y por lo tanto se podrá experimentar de manera más interesante las funciones que se desarrollarán.

1.2. El sistema operativo JOS

JOS es un sistema operativo académico utilizado en muchos cursos avanzados de sistemas operativos. A diferencia de Minix, aprovecha completamente los procesadores de 32 bits de Intel. Esto permitirá aplicar los conceptos de manejo de memoria con paginación vistos en la materia.

La cátedra preparó un paquete con una versión de JOS preparada para resolver este trabajo práctico.

1.3. Entorno para el TP

El sistema operativo JOS está pensado para ser utilizado sobre el simulador Bochs. Existen versiones de Bochs tanto para Windows como para Linux. El código de JOS deberá ser bajado de la página de la materia, ya que se le han realizado modificaciones para este trabajo práctico.

Una vez bajado el archivo `jos.tar.gz`, se lo deberá descomprimir utilizando:

```
tar xzf jos.tar.gz
```

Esto creará un directorio `jos` que contendrá los archivos de Bochs. En ese directorio bastará escribir `make` para compilar JOS, y `bochs -q` para ejecutar Bochs con el archivo de configuración provisto que carga JOS.

2. Consigna

JOS es un sistema operativo relativamente completo. Sin embargo, cierta funcionalidad debe ser completada por el alumno. Dado que no disponemos de tiempo para completar todo lo requerido, la cátedra ha resuelto las partes más tediosas.

Una estructura importante al implementar un sistema operativo, y para la interacción entre C y Assembler, es la pila (stack). El primer ejercicio del trabajo práctico requiere implementar una función que permite revisar la pila y visualizar en ella la cadena de llamadas a función.

Por otro lado, los sistemas operativos modernos se caracterizan por utilizar exclusivamnte los sistemas de paginación tanto para administrar la memoria como para implementar la protección usuario/kernel. Por esta razón el trabajo práctico requiere completar las partes de JOS que administran la memoria del sistema operativo utilizando paginación.

2.1. La pila

En esta parte del trabajo práctico se implementará una función que permite a la consola del kernel imprimir un "backtrace" de la pila: una lista de los Instruction Pointers guardados en la pila que se puede utilizar para determinar el contexto en el que se llamó a una función de código C, incluidos los parámetros que recibió.

En los programas en C, los registros ESP (stack pointer) y EBP (base pointer) tienen un significado especial. ESP apunta al último valor apilado en la pila, después del cual la memoria está libre. Recordemos que en la arquitectura IA-32 la pila crece hacia las direcciones de memoria bajas. EBP, por su parte, se asocia con el stack por una convención de software. Cada vez que se llama a una función en C, se guarda el valor de EBP en la pila. Luego, se copia el valor de ESP en EBP. De esta manera cada función dispone de una pila "propia", que se denominará *stack frame*. Los valores recibidos como parámetros se encuentran antes de EBP, y las variables locales después. Además, si todas las funciones siguen este comportamientos se puede examinar la pila para determinar la cadena de llamadas generada por la ejecución actual de un programa y los parámetros recibidos por cada función.

La consola del kernel de JOS, a la que llamaremos *monitor*, posee una función `mon_backtrace` que imprime un backtrace de la pila al ejecutar el comando `backtrace` en el monitor. Esta función se encuentra definida en el archivo `kern/monitor.c`. Se debe completar esta función para imprimir un backtrace con el siguiente formato:

```
Stack backtrace:
  ebp f0109e58  eip f0100a62  args 00000001 f0109e80 f0109e98 f0100ed2 00000031
  ebp f0109ed8  eip f01000d6  args 00000000 00000000 f0100058 f0109f28 00000061
  ...
```

La primera línea corresponde a la función que se está ejecutando en ese momento, que siempre va a ser `mon_backtrace`. La segunda línea corresponde a la función que llamó a `mon_backtrace`, y así. Se deben imprimir todos los stack frames, sabiendo que el stack se inicializa en la dirección `bootstacktop` y crece hacia abajo.

En cada línea el valor de EBP indica la base de la pila utilizada por esa función, es decir, luego de que el inicio de la función guardara el EBP anterior y copiara ESP a EBP. Dado que no hay una manera de saber, mirando solamente el stack, cuántos argumentos se le pasaron a cada función, se deberá imprimir en cada caso los primeros cinco argumentos de cada función.

2.2. Manejo de memoria utilizando paginación

Utilizando la terminología de la arquitectura IA-32, una dirección *virtual* es un par "segment:offset" antes de que sea utilizado el sistema de segmentación. Una dirección *lineal* es lo que se obtiene del sistema de segmentación, pero antes de utilizar el sistema de paginación. Finalmente, una dirección *física* es lo que se obtiene del sistema de paginación y eventualmente se envía a la memoria.

El kernel de JOS se linkea asumiendo que va a correr en la dirección lineal `0xf0100020`, pero se carga en la dirección física `0x00100020`. Al inicio, el kernel utiliza el sistema de segmentación para realizar esta transformación, pues la paginación requiere estructuras de datos que todavía no fueron creadas. Una vez que el sistema está iniciado se dejará de utilizar segmentación y todo el manejo de memoria, incluido esta transformación, será realizado utilizando paginación.

JOS divide el espacio lineal de 32 bits del procesador en dos partes. Los procesos de usuario estarán ubicados en la parte baja, mientras que el kernel estará ubicado en la parte alta. Esto implica que se usarán permisos a nivel de paginación para aislar la parte del espacio de memoria del kernel de la parte del espacio de memoria del usuario. La estructura del espacio de memoria está descripta en el archivo `inc/memlayout.h`.

Además de configurar la unidad de manejo de memoria del procesador, para que la traducción de direcciones virtuales a direcciones físicas sea correcta, el sistema operativo debe además saber qué partes de la memoria están libres y cuáles están en uso para distintos propósitos. JOS maneja la memoria en bloques de 4 KB, que corresponden al tamaño clásico de una página en la arquitectura IA-32. La memoria física se divide entonces en bloques de 4 KB que se llamarán páginas, algunas de las cuales estarán mapeadas en distintas partes del espacio virtual de direcciones, algunas incluso en más de una parte a la vez. El total de páginas se mantiene en el arreglo `pages`, que es un arreglo de estructuras `Page`, cada una de las cuales representa una página. Dado que una página puede estar mapeada en varias direcciones virtuales, la estructura `Page` incluye un contador de referencias. Las páginas libres se mantienen en la lista enlazada `page_free_list`. La lista enlazada se maneja con los macros y funciones definidas en `inc/queue.h`.

El trabajo práctico requiere completar las siguientes funciones en `kern/pmap.c`:

`page_alloc()` Asigna una página libre, sacándola de la lista de páginas libres pero sin mapearla en ninguna dirección virtual.

`page_free()` Libera una página, agregándola a la lista de páginas libres. Una página que se libera debería tener su contador de referencias en 0.

`pgdir_walk()` Recorre el árbol de páginas buscando la entrada correspondiente a una dirección virtual dada. Abstrae de esta manera la estructura de dos niveles del árbol de páginas en la arquitectura IA-32.

`page_insert()` Mapea una página en una dirección virtual dada.

`page_lookup()` Devuelve la página mapeada en la dirección virtual dada.

`page_remove()` Elimina el mapeo de una página en una dirección virtual.

JOS verifica estas funciones al terminar su secuencia de inicio utilizando la función `page_check()`. Esta función deberá ejecutar correctamente.

3. Entrega

Se deberá entregar un informe describiendo la implementación de las funciones pedidas, detallando su funcionamiento por medio de seudocodigos y explicación. El informe deberá contener el código fuente comentado de todas las funciones. Además, se entregará el código fuente de JOS modificado en soporte electrónico

4. Referencias

JOS `http://www.scs.stanford.edu/05au-cs240c/`

Bochs `http://bochs.sourceforge.net`