DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

subtitulo del trabajo

Organización del Computador II

Segundo Cuatrimestre de 2014

Integrante	LU	Correo electrónico
Nombre	XXX/XX	`mail`
Nombre	XXX/XX	`mail`

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Resumen

En el presente trabajo se describe la problemática de ...

Índice

1. Objetivos generales **3**

2. Desarrollo **3**

2.1. Segmentación y manejo de excepciones 3

2.2. Paginación 3

3. Conclusiones y trabajo futuro **3**

1. Objetivos generales

El objetivo de este Trabajo Práctico es ...

2. Desarrollo

TODO: agregar introducción a lo que hicimos?

2.1. Segmentación y manejo de excepciones

De acuerdo a lo indicado en el enunciado, definimos segmentos de código y datos en la GDT a partir del indice 4, un par con privilegios de kernel y otro con privilegios de usuario. Direccionamos los primeros 878MB de memoria con estos segmentos. Para ello, establecimos la base de cada segmento en la dirección 0x0 y, para poder representar el número en 20 bits, calculamos el limite de cada uno en bloques de 4KB. Adicionalmente, definimos un segmento de datos con privilegios de kernel para la memoria de video, basado en la dirección 0xB8000. Dado que las dimensiones de la pantalla son 80x50 caracteres y que para representar cada uno se necesitan dos bytes, el tamaño de este segmento es de 8000 Bytes. En función de esto definimos el limite del segmento en 7999, su ultimo byte direccionable.

Resolvimos el manejo de excepciones definiendo los primeros veinte indices (de 0 a 19) en la tabla de descriptores de interrupción. Para ello utilizamos un macro en el cual referimos cada entrada N al segmento de código de kernel, con el offset correspondiente su rutina de atención _isr$N$. En cuanto los atributos de cada descriptor, dejamos en 1 el bit de presencia, asignamos nivel de privilegios de kernel y usamos el tipo *interrupt*.

Para atender excepciones escribimos rutinas que muestran sus mensajes de error por pantalla. Definimos cada _isr$N$ con un macro en el cual obtenemos y mostramos el N-ésimo mensaje de un arreglo donde los guardamos previamente.

2.2. Paginación

Siguiendo las indicaciones del enunciado, mapeamos las direcciones *0x000000* a *0x3FFFFF* usando *identity mapping*. Para lograrlo en primer lugar inicializamos los primeros 4KB de memoria a partir de la dirección *0x27000* con ceros, donde luego definimos el directorio de páginas del kernel. Ubicamos la primer página del directorio en la dirección *0x28000*, con atributos de lectura y escritura, nivel de privilegios cero y el bit presente activo. Luego, para mapear el sección de memoria pedida, llenamos la tabla con las direcciones de los primeros 1024 bloques de 4KB de memoria física. De esta manera definimos la última entrada de la tabla con la dirección base 0x3FF000, haciendo direccionables los siguientes 4096-1 direcciones. Al igual que la definición de la tabla de páginas en el directorio, cada página fue definida con atributos de lectura y escritura, privilegios de kernel y bit presente activo.

Teniendo armado un directorio de páginas con una tabla de páginas definida, habilitamos paginación moviendo la dirección del directorio a CR3 y levantando el bit correspondiente en CR0.

Para el manejo de páginas libres utilizamos el puntero `proxima_pagina_libre` que denota la dirección de la próxima página libre, al que inicializamos apuntando al inicio del área libre, en la dirección *0x100000*. Administramos este puntero con la función `mmu_proxima_pagina_fisica_libre`, que devuelve un puntero a la proxima página e incrementa el valor de `proxima_pagina_libre` en 4KB.

TODO: Explicar ej. 4

3. Conclusiones y trabajo futuro