Resumen

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

Índice

Figura 1: Descripcion de la figura

1. Objetivos generales

El objetivo de este Trabajo Práctico es ...

2. Contexto

Titulo del parrafo Bla bla bla bla. Esto se muestra en la figura 1.

```
struct Pepe {

    ...

};
```

3. Enunciado y solucion

3.1. Ejercicio 1

En el ejercicio 1 se completa e inicializa la GDT, con cuatro segmentos, 2 de codigo(uno del kernel y el otro del usuario), y dos de datos(uno del kenrnel y otro del usuario), dirrecionando los primeros 878MB de memoria.(Se dejan las primeras 3 posiciones de la gdt libre, una or ser la nula y las otras dos por reestricciones del tp).El primer indice que deben usar para declarar los segmentos es el 4(contando desde 0).

Para completar la gdt, agregamos los descriptores de segmento a la GDT, modificando el archivo gdt.c. Alli, describimos los segmentos completando structuras de descriptores y descriptor de gst(str_gdt_entry, y str_gdt_descriptor, definidas en gdt.h), Alli, al nulo se le pone todo 0 y a los otros se le pone: Se las flatea, poniendoles a todas la misma direccion base (0x00) y de limite se coloca el tamano-1 /0x400 +0x3FF(se pone asi, porqu en realidad vamos a poner el de granularidad en 1, para que la cuenta nos de el tamano pedido menos 1. el tipo es read/write(0x02) en los de data, y (execute/read) en los de codigo. El s es 1 x ser de datos o codigos, la dpl es 3 o 0 dependiendo si es un segmento de usuario o del kernel. el p es 1, (???), el l esta en 0 x estar en 32 bytes. lel db esta en 1 por el mismo motivo. Tambien hay un segmento de video(del kernel) cuya direccion de entrada esla pedida por la catedra(y limite de acuerdo al tamano de la pantalla pedida.)), el resto es como un segmento de codigo definido antes. Utilizando estos defines

(imagen de toma de pantalla)

se dispuso asi en la gdt.c.

(imagen de toma de pantalla) (imagen de toma de pantalla) (imagen de toma de pantalla)

En el kernel.asm, se pasa a modo protegido, para hacerlo se pone la directiva BITS 16, (para que el linker sepa que se interpreta en 16 bits las direcciones) Luego , se desabilitan ls interrupciones(cli), se cambia el modo de video(interrupcion)(se va a modo 3h y luego se setea) Se da mensaje de bienvenida, pero como no es parte del ejerciocio no lo describo(por ahora) Se habilita la A20 (con una funcion) y se carga la gdt con la funcion lgdt(y la direccion de la tabla(y tam))/ Luego se pasa a modo protegido seteando el bit PE del registro CR0. y se salta a la siguiente instruccion, (desde seg de codigo de la gdt), que es la siguiente (en el medio, se usa la direcctiva BITS 32, para que reconozca que trabajamos con 32 bites). Se establecen los selecotores de segmento (ds, es, fs, gs, ss)(todos apuntan al segmento de datos del kernel, porque es el codigo del kernel) Con estos segmentos seteados, podemos establecer la pila moviendo la base a los reg ebp y esp (bdireccion pedida por catedra) Se imprime otro mensaje(luego describo), y se inicializa la pantalla, para ell llamamos a la funcion inicializar_pantalla, que pushea ds, mueve eax al segmento de video,(y a la primera posicion), y mientras el contador ecx loopee(tiene la cantidad de words a pintar de gris), avanzamos eax y por cada 2 bytes llenamos el lugar del color querido (01110000b ; 0111 = grey sin bright, 0000 = black sin bright). Luego popeamos ds y salimos. (porque dejamos el espacio???)

Terminamos ejercicio 1

3.2. Ejercicio 2

Tenemos que completar las entradas de la IDT para asociar las diferentes rutinas a todas las llamadas del procesador. La IDT (Interrupt Descriptor Table), es la que almacena los decriptores de interrupciones(desriptores de sistema que pueden ser de tres tipos, trap, interrupt o task, nosotros vamos a usar interrupt solamente). Para completar la IDT llamamos desde el kernel.asm a idt_inicializar.Esta funcion contiene los descriptores que seran puestos en la GDT (lo que hacemos utilizando la instruccion lidt [IDT_DESC], donde IDT_DESK el limite y la base de la IDT.) En las entradas de la IDT(IDT_ENTRY), colocamos el offset de la direccion de la rutina de atencion de interrupciones (definidas como isr(numero de interrupcion)), y las rutinas de interrupcion consisten en imprimir infinitamente el texto el nombre de la excepcion que el procesador genero, en la pantalla(en la parte superior derecha); en los bits correspondientes del descriptor de interrupciones, pusimos los atributos (presente, le dimos prioridad 0 porque son excepciones del procesador, aclaramos el tipo(interrupt) y que sea de 32bits)EL segsel es el 0x18 por ser el tercero de la gdt, donde esta el descriptor del segmento donde se encuentran las rutinas de atencion de interrupciones. Cuando se produce una interrupcion, el procesador busca en la istr la idt, va a la Idt y se va a la puerta de esa interrupcion, se usa el segmento para ir al segmento en la gdt donde esta el segmento, se le suma a la base el offset de la interrupcion para ir a la rutina correspondiente, y se la ejecuta.

Terminamos el ejercicio 2

3.3. Ejercicio 3

Vamos a limpiar el buffer de video para que se vea como lo indica la figura 7. Para ello, creamos la funcion imprimir_pantalla en screen.c. La funcion imprimir_pantalla dibuja los bordes (negro), los marcadores(la parte roja y azul de la pantalla correspondiente a cada jugador), los relojes(que luego de deberemos actualizar para que respondan a la interrupcion que genere una tarea siendo ejecutada), e imprime las vidas. Aqui tambien se utiliza la funcion print_int_sinattr, que como el nombre lo indica imprime en el lugar buscado del buffer de video el numero sin atributos,(agarramos, los que estan en actualizar vidas.)Luego tendremos que hacer esto actualizando las vidas con interrupciones. Cada vez que imprimimos excepto esta usamos la funcion print dada. Para la medicion de los margenes utilizamos defines(fijarse en la seccion defines)

Ahora vamos a definir la estructura de paginacion, y posteriormente, vamos a activar la paginacion. (esto pertenece al ejercicio 4, pero lo ponemos aca) Para definir lo primero, se inicializa el manejador de memoria llamando a mmu_inicializar,que nos coloca en las posicion del principio de las paginas libres(0x10000).A partir de ahora, las paginas seran dadas desde esta direccion de memoria. Luego, inicializamos el directorio del kernel. Lo que hacemos es (mediante mmu_inicializar_dir_kernel) es: Primero, nos guardamos la direccion del directorio de paginas del kernel(0x27000) y la direccion de las tablas de paginas (0x28000). Luego, para el numero de entradas del directorio paginas que requerimos para el identity mappig de nuestro tp (lease, una), se completa las entradas del directorio con la direccion de la tabla de paginas definida anteriormente y los atributos de estar presente y de poder ser leida y/o escrita(los ultimos bits en 3).Luego, como necesitamos todas las paginas que nos provee una tabla de paginas,(1024), llenamos cada entrada con la direccion que queremos mapear(0x00000000 a 0x003FFFFF), y los atributos de ser leida a la entrada, escrita y presente). Luego avanzamos ptabla la cantidad de entradas de la pagina.(en este caso no se necesita, pero si se requisiese otra tabla de paginas en el directorio, que el ptabla apunte a la direccion de la proxima tabla.) Luego, se completa las entradas no validas del direcctorio de paginas con 0 para que no haya problemas. Por ultimo, como vamos a cambiar la cr3 para inicializar la paginacion, flusheamos la tlb.(para invalidar chache de las traducciones default) Luego movemos la direccion del directorio cargada en cr3, y habilitamos la paginacion seteando el bit 31 de cr0.

Imprimimos el nobre de grupo usando la funcion imprimir_texto.

Terminamos el ejercicio 3

3.4. Ejercicio 4

En este ejercicio vamos a armar las funciones necesarias para construir la estructura de paginacion de una tarea.

Para ello implementamos la funcion mmu_inicializar_dir_tarea, que funciona de la siguiente manera:

Los parametros pasados en la funcion son la direccion donde esta el codigo de la tarea, y la direccion que el jugador le pasa para mapear ese codigo. Dentro de la funcion, se declara un puntero llamado pdirectorio, al cual se le pasa la direccion de la proxima pagina fisica libre(dada por la funcion mmu_-proxima_pagina_fisica_libre()). Esta sera la direccion del directorio de paginas de la tarea.(****)Tambien se declaran int que usaremos de contadores(i, j) y la el puntero pcodigo_destino, que apunta a la direccion logica 0x08000000, donde se ubicara el codigo de una tarea al ser inicializada(donde se encuentra el codigo destino). Luego, mientra i sea menor a la cantidad de paginas de identity mapping (que es 1, en este ejercicio), se declara otro puntero ptabla, (que tambien recibe la proxima pagina libre), que sera la direccion de una de las tablas de paginas del directorio de la tarea. En la direccion apuntada por pdirectorio en esa posicion(i) se declara el puntero a la tabla,(las 20 bits + altos) con los atributos para escribir, estar presente y ser del kernel.(se usar ors con defines.) Luego, usando la j como contador, rellenamos las posiciones de la tabla pasada en esa entrada del directorio con las posiciones donde deberan empezar las paginas(vamos aumentando * el tamano de pagina cada vez para que no se solapen, y la direccion de la primera pagina de la tabla es la siguiente a la de la ultima pagina de la anterior tabla que estaba en la entrada anterior del directorio de la tarea. Ademas, le pponemos los atributos de ser presente y poder ser escrita, leida, estar presente y privilegio de supervisor.(sumandole 3) Completamos el resto de las entradas del directorio de tablas con 0. Ahora ya completadas las paginas, copiar el codigo a la posicion del mapa que le indica el jugador, y copiar las paginas a partir de la direccion virtual dada. Para eso utilizamos la funcion mmmu_mapear_pagina(DIR_LOG_CODIGO_TAREA, (unsigned int) pdirectorio, dirmapa, PG_USER — PG_WRITE) para mapear la direccion virtual donde se encuentra el codigo de la tarea en la direccion del mapa pasada por el usuario, con los atributos de escribir y ser de usuario. (utilizamos como direccion de cr3 al puntero de directorio de la tarea.). Hacemos lo mismo con la pagina virtual siguiente (que es la pagina mapeada para que la tarea pueda modificarla) Por ultimo, para copiar el codigo de tareas, se mapea una direccion virtual libre a la direccion del mapa pasado, con permisos de supervisor, y el rcr3 actual, usando mmu_mapear_pagina(DIR_LOG_CODIGO_TAREA - PAGE_SIZE, rcr3(), dirmapa, PG_KERNEL — PG_WRITE); Se copia en la pagina virtual que indica el enunciado el contenido de las pagina de codigo pasado por parametro(llendo desde i en 0 hasta el final de la pagina(tam de pagina-1). Por ultimo se desmapea la pagina auxiliar que fue utilizada para que el kernel copiara a la pagina usando mmu_unmapear_pagina(DIR_LOG_CODIGO_TAREA - PAGE_SIZE, rcr3()), donde, DIR_LOG_CODIGO_TAREA - PAGE_SIZE es donde habia mapeado la pagina (0x08000000 - 0x1000) y rcr3 el cr3 (puntero). Se flushea la tlb con tlbflush para invalidar la tabla de traducciones, y se devuelve pdirectorio, (efectivamente, la direccion que el cr3 deberia usar para ir al directorio de paginas de la tarea)

Las funciones mmu_mapear_paginas y mmu_unmapear_paginas funcionan de la siguiente manera:

mmm_mapear_paginas recibe de parametros la direccion virtual(virtual), el cr3, la direccion fisica (las tres direcciones son pasadas como unsigned int), y ademas los atributos que vamos a pasar a la pagina mapeada(como unsigned char). Declaramos un puntero pdirectorio, al cual le pasamos el cr3(), y otro que llamamos ptabla. y luego, si la entrada del directorio(apuntada por el pdirectorio) de paginas solicitada por el virtual(los primeros 10 bits de virtual) no esta presente, entonces se pide la direccion de la proxima pagina fisica libre y se la asinga a ptablas. A su vez, se coloca ptablas en la entrada del pdirectorio donde dio que no estaba, (con los atributos de usuario, presente y escribible), y se llena el contenido de la tabla de paginas apuntada por esta ptabla de 0. Si por el contrario esta presente la entrada, ptabla le ponemos la direccion que de la entrada, y si no esta presente la entrada de la tabla de paginas pasada en virtual, se coloca en esa entrada la direccion de la pagina fisica pasada (efectivamente, ponienedola como direccion destino de la paginacion de la direccion virtual dada), junto a los atributos pasados y el atributo de presente. Por ultimo flusheamos la tlb para invalidar la tabla de cache.

mmu_unmapear_paginas recibe de parametros la direccion virtual(virtual) y el cr3 (no recibe los otros dos, porque como la pagina esta mapedad, no los necesita) volvemos a declarar un puntero al cual le asignamos el cr3, y una ptabla. Si la entrada del directorio de paginas apuntado por pdirectorio que

es pasada por virtual esta presente (que es cuando hay que unmapearla), entonces a ptabla apunta a la direccion de la tabla de paginas almacenada en esa entrada. Ponemos la entrada de la tabla de paginas apuntada por los bits 21-12 de virtual en 0 (no mas presente); y para i desde 0 hasta las entradas de la tabla, si no esta presente una paginase sigue, y si no esta presente ninguna pagina en ninguna entrada de la tabla, se pone la entrada en la entrada donde esta la tabla de paginas en 0 (se la unmapea).

Terminamos el ejercicio 4 (falta punto d....preguntar.)

4. Ejercicio 5

En este ejercicio vamos a completar la IDT para que pueda asociar rutinas a la interrupcion de reloj, de teclado y la de sofware 0x66 Para hacerlo utilizamos la funcion idt_inicializar que habiamos usado antes, expandiendo la funcion para ahora incluir a las interrupciones de reloj, de teclado y a la de sofware 0x66. Para las interrupciones de reloj y teclado, utilizamos el mismo procedimiento que con las de excepciones, declarando una GDT_ENTRY, (con 32 para el reloj y 33 para el teclado), cuyas rutinas son descriptas en la isr de ese numero (cuya direccion la GDT_ENTRY acomoda para que formar el descriptor deseado) Estas dos interrupciones tienen atributos 0 como las anteriones, por lo cual se usa esa funcion. La interrupcion de sofware sin embargo, como su nombre lo indica, no es del mismo nivel que las de teclado o reloj, por lo cual utilizamos declaramos una IDT_ENTRYUSR(102), cuya declaracion tiene la misma forma que una GDT_ENTRY, con la diferencia de que elnivel de privilegio es 3 en lugar de 0. Su rutina de interrupcioon tambien esta en la isr de ese numero.

Ahora vamos a describir el funcionamiento de las rutinas de interrupcion de teclado, reloj y 0x66.

5. Defines

6. Conclusiones y trabajo futuro