Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Trabajo Práctico Nro 3
System Programming - TronTank

Organización del Computador II

Primer Cuatrimestre de 2014

Grupo: **Colombia - Arepa**

Apellido y Nombre	LU	E-mail
Cisneros, Rodrigo	920/10	rodricis@hotmail.com
Rodríguez, Agustín	120/10	agustinrodriguez90@hotmail.com
Tripodi, Guido	843/10	guido.tripodi@hotmail.com

Contents

1 Introducción

En el siguiente informe se describen los módulos implementados que constituyen el código del Trabajo Práctico Nro 3 *TronTank*. Cada módulo descripto incluye una breve descripción de las decisiones de diseño tomádas por el grupo con respecto al procesador *Intel* y sus reglas de desarrollo y, de ser necesario, una explicacion de la implementación. Esto incluye en el TP: configuración de la GDT, pasaje a modo protegido, configuración de la IDT, paginación, TSS y la organización del scheduller.

2 Desarrollo y Resultados

2.1 Ejercicio 1. GDT

2.1.1 Global Descriptor Table

Como ya sabemos, el procesador inicia en "modo real", el cual direcciona a 1 MB de memoria y no posee niveles de protección ni privilegios.
Por eso necesitamos que el procesador pase a "modo protegido", para direccionar a más memoria y poder manejar distintos niveles de protección. Nuestro kernel se encargará de hacer esto.

Antes de iniciar en modo protegido, es imprescindible tener bien configurado la Tabla de Descriptores Globales, la cual contiene a los descriptores de segmento, con el fin de definir características de varias áreas de la memoria.
En el enunciado se piden una segmentacion flat, con 4 segmentos que deben direccionar a 1.75 GB: 2 para código de nivel 0 y 3 respectivamente, y 2 para datos, de nivel 0 y 3 también.

La estructura de un descriptor de segmento es la siguiente:

- L — 64-bit code segment (IA-32e mode only)
- AVL — Available for use by system software
- BASE — Segment base address
- D/B — Default operation size (0 = 16-bit segment; 1 = 32-bit segment)
- DPL — Descriptor privilege level
- G — Granularity
- LIMIT — Segment Limit
- P — Segment present
- S — Descriptor type (0 = system; 1 = code or data)
- TYPE — Segment type

Para definir los segmentos que nos requieren, los items importantes son:

- BASE: este parametro indica el comienzo del segmento. En los 4 casos, este fue 0 ya que se pidió una segmentacion flat.
- P: Present, este parametro indica si el segmento está presente en la memoria. El valor en los 4 segmentos es 0x01 ya que efectivamente estaban presentes.
- DPL: Nivel de privilegios del descriptor. Dado que se piden dos segmentos de código y dos de datos nivel 0 y nivel3, este parametro varía según cual de estos queremos implementar. Nivel 0 implica DPL = 00b y nivel 3 implica DPL = 11b.
- G. Granularity. Este flag indica si el tamaño del descriptor es mayor o menor que 1 Mb. Esto sucede dado que solo se poseen 20 bits para indicar el tamaño del segmento. En particular, si G = 1 entonces el valor de los 20 bits será multiplicado por 4 Kb provocando que con solo 20 bits pueda representar 4Gb de memoria. En nuestro caso queremos un tamaño de 733MB entonces necesitamos G = 1.
- Limit: Tamaño del segmento. Va para los 4 segmentos lo mismo.
  El valor es: 56575
  $56575 = 0xDCFF$

- Type: Indica si es un segmento de código o de datos. Para el segmento de código de nivel 0 ponemos el valor de 0x08, indicando que es "Execute only". para el segmento de código de nivel 3 se usa 0x0A, *Read / Execute*. Mientras que para los 2 de datos ponemos el valor de 0x02, indicando que son de Read/Write.

También se define un segmento que reservado para el área de la pantalla en la memoria. Sabemos que empieza en la dirección base 0x000B8000, con un tamaño de 0x0FA0. Dado que se utilizará como un segmemto de datos, su tipo es de Lectura/ Escritura.

También necesitamos entradas para cada una de las tareas. Es decir, selectores de TSS. Estos serán definidos de forma dinámica y no hardcodeados, basándose en la posición de su respectivo TSS. Básicamente cada una de estas entradas de la GDT para las TSS fue inicada de la siguiente manera:

- BASE: Dirección donde fue definido el comienzo de la TSS para cada respectiva tarea.
- P: Present. Este flag debe ir seteado para todas las TSS.
- DPL: las tareas corren en nivel 3, por lo tanto, el DPL = 3 salvo para las tareas INICIAL e IDLE que deben correr en nivel 0.
- limit: Como mínimo las TSS tienen un tamaño de 104 bytes es decir 0x67. Esto es como mínimo ya que existe la posibilidad de extender el IO Map Base Address.
- type: este es particular. dado que es un tipo de descriptor de segmento, el valor tiene que ser 0x09.
- S: este flag determina si el descriptor se refiere a un segmento de código o datos o si es de sistema. En este caso como los descriptores de TSS son de sistema S = 0.

2.1.2 Pasaje a modo protegido

En función de pasar a ejecutar en modo protegido el manual de *Intel*[1] explicita una serie de pasos que se deben seguir para complir con esto.

- Habilitar A20. al realizar esto habilitamos el acceso a direcciones superiores a 1 Mb de memoria.
- Una vez que tenemos configurada la gdt, guardamos su ubicacion en una variable gdt_desc. Para que luego la instrucción lgdt pueda cargar la direccion de comienzo de la GDT.
- Seteamos el flag PE del registro CR0, que indica "Protected Envirnoment".
- Por último para pasar a modo protegido hacemos un jmp al comienzo del segmento de código de nivel 0.
- Una vez ahí acomodamos todos los segmentos apuntando a datos de nivel 0 y seteamos la pila del Kernel en 0x27000 según lo indicado por el enunciado.

[1] Ver Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3 System Programming Guide

2.2 Ejercicio 2 IDT

2.2.1 Interrupt Descriptor Table

A través de la IDT, definimos donde está el código de las interrupciones que manejaremos.
La estructura de una entrada en la IDT está definida en idt.h y en idt.c son iniciadas todas las entradas. Por medio de una macro se cargan las primeras 19 interrupciones del procesador, que van desde la división por 0 hasta la interrupción SIMD.
Luego son completadas todas las entradas restantes de la tabla con entradas de interrupciones inválidas con el propósito de manejar de alguna forma todas las interrupciones posibles. Algunas de estas son definidas nuevamente:

- Interrupción 0x32: Clock.
- Interrupción 0x33: Teclado.
- Interrupción 0x52: Servicios del sistema (syscalls).

En isr.asm se encuentra el código donde atendemos estas interrupciones. Saliendo de las 3 interrupciones mencionadas arriba (clock, teclado, syscall), todas las interrupciones seran atendidas de una forma similar (para esto usamos un macro). Se realizan las escrituras pertinentes en pantalla y despues se desalojara la tarea que la causo. Es importante notar que no todas las interrupciones se imprimen igual, pues algunas traen opcode, asi que en pantalla tenemos un array que nos indica cuales instrucciones tienen opcode y cuales no.

La estructura de una entrada de la idt, definida en idt.h, es la siguiente:

- offset_0_15: primeros 16 bits del offset al entry point, que atenderá la interrupción
- segsel: selector de segmento de codigo de nivel 0 la gdt
- attr: atributos de la entrada: Present, DPL, D. Esto varían según si la interrupción es de Reloj o Teclado que llevan DPL = 00b o Servicios cuyo DPL = 11b.
- offset_16_31: segundos 16 bits del offset al entry point.

Indice	Descripcion	P	DPL	D
0...19	Ins del procesador	1	0	1
32	Clock	1	0	1
33	Teclado	1	0	1
52	Syscall	1	3	1

2.2.2 Proceso para activar interrupciones

Para poder activar todas estas interrupciones y sus respectivos handlers se siguen los siguientes pasos:

- Mediante el uso de la instrucción LIDT [IDT_DESC], cargamos el principio del array donde tenemos cargados todas las interrupciones
- Por último se deshabilita, se resetea y se vuelve a habilitar el pic que obtiene las interrupciones.[2]

[2]Las funciones de deshabilitar, habilitar y resetear fueron provistas por la cátedra.

2.3 Ejercicio 3. Paginación

2.3.1 Kernel, Identity Mapping

Debemos mapear con Identity mapping las direcciones 0x00000000 a 0x00DC3FFF. Para esto fueron necesarios:

- 1 Tabla de Directorios de páginas que empieza en la direccion 0x27000.
- 4 Entradas de tabla de directorios.
- 4 tablas de páginas. La 3 primeras Page table posee sus 1024 entradas completas y tienen como base la dirección 0x28000, 0x30000,0x32000 respectivamente. Y la cuarta tabla con 452 entradas presentes.
  De esta forma conseguimos mapear todas las direcciones

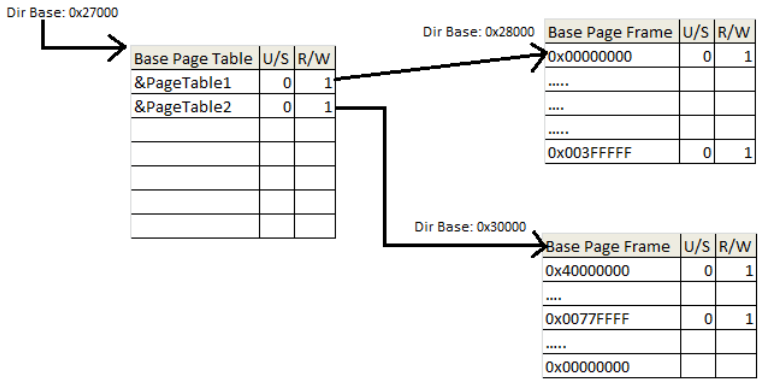
Las entradas de directorio para Kernel son cargadas de la siguiente manera[3]:

- P = 1.
- R/W = 1.
- U/S = 0.
- Direccion de la Page Table = 0x28000,0x30000,0x32000 según corresponda la page table.

Las entradas de Page Table para el Kernel son cargadas de la siguiente manera[3]:

- P = 1.
- R/W = 1.
- U/S = 0.
- Direccion del Page Frame desde 0x00000000 a 0x00DC3FFF según corresponda.

A continuación se detalla un esquema para una mejor comprensión de lo explicado:



2.3.2 Activación de paginación

Luego de armar el directorio de páginas podemos habilitar la paginación. Para esto seguimo los siguientes pasos:

- Cargar en CR3 la direccion al inicio del directorio de páginas.
- Setear el bit mas significativo del registro CR0.

[3]Se pueden considerar a los flags no declarados como no seteados, es decir, iguales a 0.

2.4 Ejercicio 4. Paginación de tareas

2.4.1 Paginación de las tareas

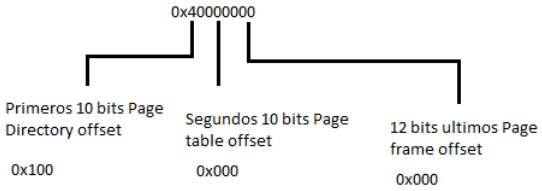
Para la paginación de tareas se necesitaron los siguientes módulos por cada tarea:

- Inicializar un directorio de páginas con 5 entradas, 4 para el Kernel iguales a las descriptas en el ejercicio 3 (es decir, identity mapping) y otra entrada con dos entradas de tablas presentes para direccionar a las páginas de código y pilas de cada tarea. Este Page directory está definido en la direccion 0x00100000
- Dentro de la Page Table de las tareas se encuentran definidas las entradas de cada página de la tarea.

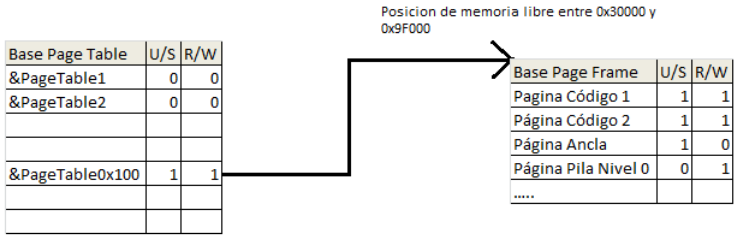
Con el fin de simplificar la cantidad de pasos, las direcciones fisicas de las paginas cada tarea estaran mapeadas en arrays externos. De esta forma no tendremos que buscar dentro de la GDT para buscar una direccion fisica a la hora de hacer otras operaciones, que puede ser costoso en cuestiones de tiempo y dejar un codigo poco claro.

A diferencia del ejercicio anterior, como en este caso estamos mapeando tareas, estas se diferencian de las páginas de Kernel en que los attributos que utilizan son diferentes, es decir, las tareas al correr en nivel 3 requieren que sus Page Directory entries y sus Page table entries posean U/S = 1. De lo contrario no puedo acceder a esas páginas como nos lo demostroó nustra experiencia en la implementación.

El siguiente esquema explica mas simplemente lo nombrado anteriormente. Tener en cuenta que esto debe realizarse por cada tarea y que si bien cada una está mapeada al mismo lugar, la base del Page Directory es distinto para cada tarea, provocando que cada una tenga su propio mapeo.



Y Luego de esto el esquema de paginación nos quedaría algo así:



Base Page Table	U/S	R/W
&PageTable1	0	0
&PageTable2	0	0
&PageTable0x100	1	1

Base Page Frame	U/S	R/W
Pagina Código 1	1	1
Página Código 2	1	1
Página Ancla	1	0
Página Pila Nivel 0	0	1
.....		

Cabe destacar que para poder cumplir con los ejercicios siguientes, una tarea IDLE va a tener que ser mapeada al comienzo de la paginación de lo contrario, no tener una tarea mapeada me imposibilita arrancar el sistema de scheduling que será introducido mas adelante.[4]

[4] Ver sección Ejercicio 7, Scheduling.

2.5 Ejercicio 5 IDT / Clocks, Teclados y Syscalls

La siguiente sección está dedicada a los *Handlers* o manejadores de las interrupciones. Estos son los códigos que se ejecutan cuando alguna porción del systema produce un error o llama a una interrupción mejor conocida como syscall o servicios del sistema.
Nuestro Kernel cuenta con 3 interrupciones que poseen handlers. Estas fueron mencionadas en el punto 3. Procederemos a explicar cada una de ellas:

- Clock: El clock es una interrupción que se ejecuta cada ciertos ticks de reloj. La misma se encarga de buscar el siguiente selector de segmento según es especificado en la sección 7. y realizar el salto a dicho selector que puede corresponder a una tarea o la tarea IDLE.

- Teclado: La interrupción de teclado cumple la función de cambiar el estado del juego y mostrar por pantalla el contexto de la tarea segun el numero de tecla apretado. Ej. Si se presiona la tecla 1 muestra por pantalla los datos del tanque 1.

- Servicios o Syscalls: Esta interrupción brinda al systema una serie de servicios o funciones a las tareas:

  - Minar: Estas minas destruyen a los tanques que se quieran mover a la posicion donde esta la mina.
    Las minas duran una sola vez, es decir, si un tanque es destruido por una mina, esta desaparece (tambien es destruida). Las minas tambien pueden ser destruidas si un misil cae sobre la misma.
    El unico parmetro de este servicio es la posicion donde colocar la mina. Un tanque puede colocar una mina en alguna de las 4 posibles direcciones a su alrededor de la posicion actual donde se encuentra.
    Una vez llamado este servicio el scheduler desalojara al tanque y ejecutar a la proxima tarea mientras se encarga de colocar la mina anti-tanque en su lugar.

  - Lanzar Misil: Los tanques tienen cañones que les permiten disparar balas a sus enemigos, ahora, como los disparos seran dirigidos a posiciones precisas del mapa, los llamaremos misiles.
    Dicho esto, este servicio permitir a lanzar misiles a posiciones de el mapa direccionadas de forma relativa a la posicion del tanque. Los valores posibles para x e y no pueden superar en modulo el tamaño maximo de el mapa, es decir 50.
    El buffer donde se almacena el misil debe ser un rango dentro del area mapeada por la tarea, ademas el tamaño del mismo no debe superar los 4096 bytes.
    El servicio se debe encargar de copiar el buffer del misil dentro al principio de la pagina fisica idenfiticada por las coordenadas x e y dentro de el mapa. Una vez lanzado el misil, el scheduler dara paso a la proxima tarea.

  - Mover: El codigo de cada tanque esta mapeado incialmente en dos paginas de memoria a partir de la direccion correspondiente a los 128mb de memoria virtual. A partir de esta direccion, luego de las dos paginas mapeadas inicialmente, se iran mapeando las paginas de memoria fisica que el tanque recorra. Si a la hora de mapear una nueva pagina fisica, esta esta ya esta mapeada dentro del espacio del tanque, entonces no se mapea nuevamente.

Cabe destacar que las funciones implementadas en C para las syscalls, minar, misil y mover, se encuentran allí dado que manejan páginas de memoria haciendo que ubicarlas en mmu.c sea lo más conveniente para aprovechar todas las funciones y estructuras utilizadas.

2.6 Ejercicio 6. TSS

Para que el procesador pueda despachar, ejecutar o suspender múltiples tareas, es necesario salvar el estado de las mismas. La arquitectura provee mecanismos para esto. El segmento de estado (TSS, Task State Segment), es el que se encarga de almacenar la información del estado de una tarea.

Una tarea está identificada por el selector de segmento de su TSS. Y a su vez la TSS es un segmento, por lo tanto debe estar descripto en la GDT junto con los descriptores de segmento de código y datos.[5]

Tenemos 8 tareas y definimos un total de 2 TSS, y otro lo dejamos en blanco para la tarea inicial donde se hace el primer salto.

Al contar con dos TSS y 8 tareas realizamos cambios de contextos guardando los datos de las tareas con una estructura auxiliar, permitiéndonos así volver más tarde a esa tarea y no perder la información de la misma. El cambio de una tss a otra lo realizamos en nuestra estructura de scheduler ejecutando JMP FAR.
Por esto mismo es necesario"incializar" una TSS para que cuando entremos por primera vez la información sea válida. Al momento de inicializar estos segmentos, cada tarea tendran TSS virtualmente idénticas, con la excepción del eip y pequeños cambios con respecto a las posiciones de las pilas.

Como selectores de segmentos de GDT usamos los que definimos para las tareas (es decir, los de nivel 3), y seteamos el RPL en 0x03 para evitar un GPE.

Una de las grandes ventajas de estar trabajando con direcciones virtuales es que no tenemos que saber la dirección física exacta de cada tarea para inicializarlas. Sabemos que todas las tareas comparten ciertas direcciones virtuales, asi que seteamos el directorio de página (CR3) correspondiente a esa tarea y podemos usar direcciones virtuales idénticas para todas las tareas.

[5] Ver sección 1 para mas información sobre esto.

2.7 Ejercicio 7. Scheduller

El Scheduler es la estructura más grande y quizás más compleja de nuestro trabajo. Su funcion es simple, coordinar en que orden ocurren los eventos en nuestro Kernel y determinar ciertas acciones como si una tarea es desalojada por una operacion ilegal o mina, o un cambio simple de tareas por tick de reloj. Conceptualmente nos imaginamos al Scheduler teniendo:

1. Un timer llamado quantum que dictamina cuantos ciclos le queda a la corrida de tareas hasta que vuelva a empezar.

2. Un estado pause, que en caso de estar activo hara que corra solo la tarea Idle.

3. Un array donde guardamos el contexto de todas las tareas incluyendo la Idle.

Corrida tarea:

La interrupción de clock se encarga de realizar todos los saltos y cambios de tareas, exceptuando el salto a idle (que puede ser hecho en cualquier momento). El scheduler es la estructura que le informa hacia donde ir siguiendo. De esta forma, mantenemos el código facilmente segmentado.

Una excepción interesante es el caso en el que no querramos saltar a ningun lado sino seguir en la tarea actual. Por ejemplo, si me queda una sola tarea y estoy en la corrida de tareas aún con quantum me gustaría pertenecer en esa tarea. Para esto el scheduler devuelve el selector de segmento 0, el cual es reconocido por el clock como una instrucción para volver a la tarea anterior (iret) y no realizar ningun salto. (tratar de saltar a una tarea en uso daria error).

2.8 Screen

La estructura screen también es una parte integral del trabajo porque no solo se ocupa de mostrarle la información al usuario, en ella también se guardan ciertos datos y se interpreta cierta información. Por esto mismo nos pareció importante agregarle una sección.

Nuestra pantalla está separada en 3 gráficos distintos, el mapa, la tabla de contexto de tareas (que se encuentra en la parte derecha), la tabla de errores (que se encuentra en la parte inferior derecha) . Todas tienen su propias funciones, y la ventaja que tienen es que pueden ser chequeadas cuantas veces uno quiera presionando la respectiva tecla(Ver interrupcion Teclado).

La tabla de contexto de tareas es una estructura que imprime el estado de una tarea, es decir los registros. En caso de que la tarea muera, esto muestra el ultimo contexto activo antes de morir.

2.9 Game

Para manejar toda la lógica del juego, los cambios en la pantalla y los mapeos correspondientes tenemos una estructura llamada Game. Básicamente se encarga de las 3 posibles acciones de una tarea: minar, mover y lanzar misiles. Tiene guardado internamente una matriz tablero, donde almacenamos el estado del juego actual. Ni bien comienza el juego, procedemos a inicializarlo: ponemos todos los valores de la matriz tablero en 0 y distruibuimos las tareas a lo largo y ancho del mapa. Para indicar que el tanque está en cierta posición, en la matriz tablero ubicamos el número del tanque en la posición correspondiente.

1. **Minar:** Recibimos la posición relativa al tanque que quiere minar y calculamos la posición que le corresponde al tablero y con esa, colocamos el valor -1 en el tablero (arbitrario, indica que es una mina) y la dibujamos en pantalla.

2. **Misil:**En esta funcion, copiamos a un cierto destino dado en parte por parametro(por parametro nos dan cuanto se movera el misil de nuestra posicion actual)la porcion de codigo que nos dan por parametro. Esto, similar a lo conocido como virus, en caso de caer dicho misil en una parte del codigo de otro tanque se lo borrara o cambiara, generando un posible error en ese tanque.

3. **Mover:** Para realizar esta funcion del juego utilizamos un struct tanque, el cual tiene su posicion columna y fila, lo que hacemos es incrementar o decrementa el valor fila y/o columna segun corresponda. A su vez dentro de este struct tenemos la ultima direccion virtual mapeada, y se la aumentamos siempre y cuando se mueva a una posicion donde no estaba. En caso de haber una superposicion con otro tanque no ocurrira nada, en cambio si en esa posicion el tablero$_{[fila],[columna]}$ es igual a -1, el tanque explotara y dejara de andar.