PREG TEORICAS ORGA

TPSP
MODO PROTEGIDO

1. Explorando el manual Intel *Volumen 3: System Programming. Sección 2.2 Modes of Operation*. ¿A qué nos referimos con modo real y con modo protegido en un procesador Intel? ¿Qué particularidades tiene cada modo?

--> En los procesadores intel existen dos modos de operación principales. Por un lado esta el modo real que es el modo operativo que proporciona el entorno de programación del procesador Intel 8086, con pocas extensiones. Por otro lado está el modo protegido que es el nativo del procesador el cual por su arquitectura cuenta con flexibilidad, protección de memoria, segmentación, paginación soporte multitarea, niveles de privilegio y virtualización y la mas importante mantiene compatibilidad con versiones anteriores y más seguridad.

2. Comenten en su equipo, ¿Por qué debemos hacer el pasaje de modo real a modo protegido? ¿No podríamos simplemente tener un sistema operativo en modo real? ¿Qué desventajas tendría?

--> Hacer el pasaje a modo protegido nos permite "formatear" y modificar los comportamientos mas crudos del comportamiento de la maquina. Las desventajas de utilizar el modo real es que tiene medidas para automatizar el power-up y reset, así como protecciones de los procesos que nos impedirían crear nuestro propio kernel sumando a que solo se pueden usar registros de 16 bits y que al carecer de proteccion podria tener programas mal escritos o mal intencionados que cambien archivos o rompan el sistema en su totalidad.

3. Busquen el manual *volumen 3 de Intel en la sección 3.4.5 Segment Descriptors*. ¿Qué es la GDT? ¿Cómo es el formato de un descriptor de segmento, bit a bit? Expliquen brevemente para qué sirven los campos *Limit*, *Base*, *G*, *P*, *DPL*, *S*. También pueden referirse a los slides de la clase teórica, aunque recomendamos que se acostumbren a consultar el manual.

--> La GDT es una estructura en donde se define cada segmento del sistema dando a cada uno distintos atributos. Algunos de estos campos son:
Limit en donde se especifica el tamaño del segmento
Base donde se indica la ubicación del byte 0 del segmento
G donde se indica si el tamaño del segmento se debe interpretar en una escala de 1 byte a 1MB incrementando de a bytes (flag en 0) o si se debe interpretar en un rango de 4 KB a 4 GB, en incrementos de 4 KB (flag en 1).
P indica si el segmento esta presente o no.
DPL indica el nivel de privilegio (entre 0 y 3), controla el acceso al segmento.
S Especifica si el descriptor de segmento es para un segmento del sistema (el indicador S está desactivado) o un segmento de código o datos (el indicador S está activado).

6. En el archivo `gdt.h` observen las estructuras: `struct gdt_descriptor_t` y el `struct gdt_entry_t`. ¿Qué creen que contiene la variable `extern gdt_entry_t gdt;` y `extern gdt_descriptor_t GDT_DESC;`?

--> gdt va a ser el arreglo de descriptores de segmento que conformara nuestra GDT. GDT_DESC sera el descriptos que utilizamos como operando en la instrucciom LGDT para cargar nuestra GDT en el registro GDTR.

7. Buscar en el Volumen 3 del manual de Intel, sección 3.4.2 *Segment Selectors* el formato de los selectores de segmento. Observar en el archivo `defines.h` las constantes con los valores de distintos selectores de segmento posibles. Completen los defines faltantes en `defines.h` y entiendan la utilidad de las macros allí definidas.
  USAR LAS MACROS para definir los campos de los entries de la gdt. En lo posible, no hardcodeen los números directamente en los campos.

--> En defines.h

8. En el archivo `gdt.c`, completen en el código la Tabla de Descriptores Globales (GDT) con los 4 segmentos definidos en el punto 5. (**usen, en la medida de lo posible, los defines del punto anterior**)

--> En gdt.c

9. Ya está todo listo para cargar la GDT. Antes de eso, completen todo lo necesario en `kernel.asm` hasta "Habilitar A20" inclusive[^2]. Tengan en cuenta que las macros de impresión se encuentran definidas en `print.mac` y que para los colores hay constantes definidas en `colors.h`. Las macros reciben sus argumentos separados por comas, sin paréntesis ni nada. Por ejemplo:

  ```print_text_rm arg1, arg2, arg3, arg4, arg5.```

  Hint: investiguen para qué puede servir la instrucción **cli** en el manual 2.

--> Cambios en archivos

10. Busquen qué hace la instrucción LGDT en el Volumen 2 del manual de Intel. Expliquen con sus palabras para qué sirve esta instrucción. En el código, ¿qué estructura indica donde está almacenada la dirección desde la cual se carga la GDT y su tamaño? ¿dónde se inicializa en el código?

--> La instrucción LGDT carga los valores del operando fuente en el registro (GDTR) o registro de tabla de interrupciones (IDTR). El operando fuente especifica la ubicación de 6 bits en memoria El operando fuente especifica una ubicación de memoria de 6 bytes que contiene la dirección base (una dirección lineal) y el límite (tamaño de la tabla en bytes) de la tabla de descriptores globales (GDT) o de la tabla de descriptores de interrupciones (IDT). Si el atributo del tamaño del operando es de 32 bits, se cargan en el registro un límite de 16 bits (los 2 bytes inferiores del operando de datos de 6 bytes) y una dirección base de 32 bits (los 4 bytes superiores del operando de datos).

11. Completen el archivo `kernel.asm` en la sección de cargar la GDT usando lo averiguado en el punto 8 para cargar la GDT.

--> En kernel.asm
  ```

13. Investiguen en el manual de Intel *sección 2.5 Control Registers*, el registro CR0. ¿Deberíamos modificarlo para pasar a modo protegido? Si queremos modificar CR0, no podemos hacerlo directamente. Sólo mediante un MOV desde/hacia los registros de control (pueden leerlo en el manual en la sección citada).

--> Debemos modificar el CR0 ya que si no cambiamos a este no podemos hacer uso del modo protegido.

14. A continuación, completen la sección del `kernel.asm` escribiendo un código que modifique CR0 para pasar a modo protegido. Tengan en cuenta las averiguaciones y comentarios del punto 13.

--> En kernerl.asm

15. Notemos que a continuación debe hacerse un jump far para posicionarse en el código de modo protegido. Miren el volumen 2 de Intel para ver los distintos tipos de JMPs disponibles y piensen cuál sería el formato adecuado. ¿Qué usarían como selector de segmento?

--> Usamos el formato jmp ptr 16:32, el selector de segmento que necesitamos es aquel que apunta a la seccion de codigo nivel 0 con el offset correspondiente al inicio de nuestro codigo para el "setup" de modo protegido una vez hecho el salto

16. Dentro del modo protegido, vamos a cargar los registros selectores de segmento (`DS`, `ES`, `GS`, `FS` y `SS`). Todos se van a iniciar en el segmento de datos de nivel 0. Para esto, completar el archivo `kernel.asm` dentro del código que se ejecuta ya en modo protegido (etiqueta `modo_protegido`)

17. Setear la pila del kernel en la dirección `0x25000`. Configuren los registros que indican dónde está la base y el tope de la pila.

--> En kernel.asm

18. Impriman el mensaje de bienvenida a modo protegido e inserten breakpoint en la instrucción siguiente

--> En kernerl.asm

22. Observen el método `screen_draw_box` en `screen.c` y la estructura `ca` en `screen.h` . ¿Qué creen que hace el método **screen_draw_box**? ¿Cómo hace para acceder a la pantalla? ¿Qué estructura usa para representar cada carácter de la pantalla y cuanto ocupa en memoria?

--> Asigna a cada posición fila x columna de la pantalla un caracter con un atributo. Para acceder a este se necesita un uint32_t que represente los valores inciales de fila y columna. Para representar cada caracter de la pantalla se utiliza ca_s que esta definida en sceen.h y ocupa 2 bytes

23. Escriban una rutina `screen_draw_layout` que se encargue de limpiar la pantalla y escribir el nombre de los integrantes del grupo (o lo que deseen) en la misma en el archivo `screen.c` y llamen a dicho método desde `kernel.asm`. Pueden usar diferentes fondos, colores e incorporar dibujos si así lo quisieran.

--> En archivo

24. Resumen final, discutan en el grupo qué pasos tuvieron que hacer para activar el procesador en modo protegido. Repasen todo el código que estuvieron completando y traten de comprenderlo en detalle ¿Qué cosas les parecieron más interesantes?

-> Las cosas que nos parecieron más interesantes de esta parte fue como haciendo el cambio de modo del procesador para contar con un modo mucho más flexible , con protección de memoria y muchisimas mas posibilidades

INTERRUPCIONES:

1. En el archivo idt.h, pueden encontrar la IDT definida como un arreglo de idt_entry_t declarado sólo una vez como idt. El descriptor de la IDT en el código se llama IDT_DESC.
En el archivo idt.c pueden encontrar la definición de cada una de las entradas y la definición de la función idt_init que inicializa la IDT definiendo cada una de sus entradas usando la macro IDT_ENTRYx.

a) :pen_fountain: Observen que la macro IDT_ENTRY0 corresponde a cada entrada de la IDT de nivel 0 ¿A qué se refiere cada campo? ¿Qué valores toma el campo offset?

--> En IDT_ENTRY0 cada entrada se va a referir a:
- offset_15_0 y offset_31_16 se refieren a la dirección lógica y estas en conjunto forman una dirección de 32 bits que el procesador los junta para formar la dirección completa de la rutina.
- segsel es el selector de segmento que es el segmento de código en la GDT donde se encuentra la rutina.
- type es el tipo de entrada que nos va a indicar si estan activas o desactivas las interrupciones
- dpl (Descriptor Privilege Level) nos da a conocer el nivel de privilegio requerido para usar esta entrada.
- present nos da a conocer si la entrada esta activa y es valida (1) o si el procesador debe ignorarla (0).

Por otro lado el offset representa la dirección de memoria de la rutina de interrupción (_isrN) y toma los valores de offset_15_0 y offset_31_16.

b)Completar los campos de Selector de Segmento (segsel) y los atributos (attr) de manera que al usarse la macro defina una Interrupt Gate de nivel 0. Para el Selector de Segmento, recuerden que la rutina de atención de interrupción es un código que corre en el nivel del kernel. ¿Cuál sería un selector de segmento apropiado acorde a los índices definidos en la GDT[segsel]? ¿Y el valor de los atributos si usamos Gate Size de 32 bits?

--> El selector de segmento al que debe apuntar GDT[segsel] es GDT_CODE_0_SEL, ya que la rutina de interrupción corre en modo privilegiado (nivel 0). Si usamos Gate Size de 32 bits debe ser 14 ya que este valor le indica al procesador que la entrada es una interrupcion

c)De manera similar, completar la macro IDT_ENTRY3 para que defina interrupciones que puedan ser disparadas por código no privilegiado (nivel 3).

-> En archivo

3.¿Qué oficiaría de prólogo y epílogo de estas rutinas? ¿Qué marca el iret y por qué no usamos ret?

--> En las rutinas de interrupción, el prólogo y el epílogo son los encargados de registrar y restaurar los contextos de ejecución del programa al momento de llamar a estas interrupciones. Usamos la instrucción iret porque, a diferencia de ret, restaura no solo la dirección de retorno, sino que también los flags y los segmentos que fueron guardados automáticamente al entrar a la interrupción. No usamos ret ya que al no restaurar todo el contexto podría llevar a una falla.

PAGINACIÓN

a) ¿Cuántos niveles de privilegio podemos definir en las estructuras de paginación?
→ 2, kernel y user
b) ¿Cómo se traduce una dirección lógica en una dirección física? ¿Cómo participan la dirección lógica, el registro de control CR3, el directorio y la tabla de páginas? Recomendación: describan el proceso en pseudocódigo

→ Veamos el ejemplo dirección lógica 0x4A125515 dado en clase:

```c
directory_offset = direccion_logica[31:22]          extendido a 32 bits
table_offset = direccion_logica[21:12]    extendido a 32 bits
offset = direccion_logica[11:0]     extendido a 32 bits

page_directory_pointer = CR3 & 0xFFFFF000
page_table_pointer = page_directory_pointer[directory_offset]
page_address_pointer = page_table_pointer[table_offset]
value = page_address_pointer[offset]
```

c) ¿Cuál es el efecto de los siguientes atributos en las entradas de la tabla de página?
- D (dirty) → indica si de escribio a memoria controlada por esta pte (escrito por el procesador)
- A (accessed) --> indica si de accedio a memoria controlada por esta PTE (escrito por el procesador)
- PCD (page cache disable)--> deshabilita cachear los datos de pagina asociada
- PWT (page write through)--> deshabilita hacer write-back al escribir en la página asociada
- U/S (user/supervisor) -->determina si un proceso en modo usuario puede acceder a la memoria controlada por esta PTE
- R/W (read/write) -->determina si un proceso puede escribir en la memoria controlada por esta PTE
- P (present) → es el bit 0 (siempre en 1), indica que esta traducción sea valida

d) ¿Qué sucede si los atributos U/S y R/W del directorio y de la tabla de páginas difieren? ¿Cuáles terminan siendo los atributos de una página determinada en ese caso? Hint: buscar la tabla Combined Page-Directory and Page-Table Protection del manual 3 de Intel

→ Cuando los atributos difieren entre la entrada de directorio y la tabla de página, los atributos efectivos de acceso a una página se determina por el más restrictivo entre ambos. Para r/w solo se permitirá la entradas son r/w = 1 y y en u/s también de una no contener 0 en r/w solo se podrá leer y no escribir y en u/s solo el supervisor podrá acceder a la página.

e) Suponiendo que el código de la tarea ocupa dos páginas y utilizaremos una página para la pila de la tarea. ¿Cuántas páginas hace falta pedir a la unidad de manejo de memoria para el directorio, tablas de páginas y la memoria de una tarea?

→ En el caso dado vamos a necesitar una pagina para el directorio, otra para la tabla y par a la tarea 2 de codigo y una para la que se encuentra en la pila, entonces en total vamos a necesitas 5.

g) ¿Qué es el buffer auxiliar de traducción (translation lookaside buffer o TLB) y por qué es necesario purgarlo (tlbflush) al introducir modificaciones a nuestras estructuras de paginación (directorio, tabla de páginas)? ¿Qué atributos posee cada traducción en la TLB? Al desalojar una entrada determinada de la TLB ¿Se ve afectada la homóloga en la tabla original para algún caso?

→ El translation lookaside buffer es una caché de traducciones utilizado para acelerar la traducción de direcciones virtuales a direcciones físicas. Es importante purgarlo al modificar las estructuras de paginación ya que de lo contrario podríamos observar traducciones cacheadas(traducciones viejas).
Cada entrada en la TLB corresponde a una página y contiene:
-Dirección física correspondiente al numero de pagina virtual.
-Permisos de acceso combinados.
-Atributos de una entrada de la estructura de paginación que identifica el marco
No se ve afectada la tabla original en ningún caso ya que al ser una caché, solo hace una copia temporal, sin modificar la tabla de páginas.

b) Completen el código de copy_page, ¿por qué es necesario mapear y desmapear las páginas de destino y fuente? ¿Qué función cumplen SRC_VIRT_PAGE y DST_VIRT_PAGE? ¿Por qué es necesario obtener el CR3 con rcr3()?

→ En el código de copy_page es necesario mapear y desmapear ambas direcciones físicas a direcciones temporales ya que el modo protegido del procesador no nos permite trabajar directamente con direcciones físicas. En definitiva lo que hacemos es cómo hacer una copia temporal y desde estas accedemos al contenido en la página pero sin estar realmente en la página ni manipulandolas, una vez terminada la copia de la página devolvemos es el espacio de la copia temporal ya que ya no será utilizada. SRC_VIRT_PAGE y DST_VIRT_PAGE van a ser los espacios en donde nosotros podemos trabajar que nos permiten escribir y leer páginas físicas pero sin interferir con otras. Es esencial obtener el CR3 con rcr3() ya que para mapear y mapear vamos a necesitar tener la dirección de page directory para saber en que estructura de paginación hacer los mapeos o desvinculaciones.

TAREAS

1. Si queremos definir un sistema que utilice sólo dos tareas, ¿Qué nuevas estructuras, cantidad de nuevas entradas en las estructuras ya definidas, y registros tenemos que configurar?¿Qué formato tienen? ¿Dónde se encuentran almacenadas?

→ Si queremos definir una sistema que utilice solo dos tareas vamos a necesitar dos TSS (una para cada tarea) en memoria y dos nuevas entradas en la GDT (un descriptor de tss para cada tss), en esta también debemos chequear de tener selectores de segmento de codigo y dato disponibles para configurar estas y de no tenerlos conseguirlos. En cuanto a registros a configurar vamos a necesitar un TR que apunte a la tarea actual y el CR3 que se cargará en el momento de iniciar la tarea con el valor de paginación correspondiente.

2. ¿A qué llamamos cambio de contexto? ¿Cuándo se produce? ¿Qué efecto tiene sobre los registros del procesador? Expliquen en sus palabras que almacena el registro TR y cómo obtiene la información necesaria para ejecutar una tarea después de un cambio de contexto.

→ El cambio de contexto es el momento en el que el procesador, a raíz de una decisión por parte del scheduler fetchea el estado de otra tarea que está en estado idle para poder continuar ejecutándola. Estos cambios se pueden producir cada ciclo de clock. Los registros del procesador se almacenan para la tarea que estaba siendo ejecutada hasta el context switch (esto se hace usando el registro TR para buscar el TSS) y luego toma los valores de los registros de la nueva tarea y los carga para resumir la ejecución. El registro TR almacena un selector de segmento de estado de tarea, la dirección base y el tamaño del segmento (el selector es buscado en la GDT para obtener el segmento de estado de tarea y con eso se obtiene el límite), el TR tiene una parte invisible (sólo visible para el procesador) que usa para cachear la base y límite del TSS.

3. Al momento de realizar un cambio de contexto el procesador va almacenar el estado actual de acuerdo al selector indicado en el registro TR y ha de restaurar aquel almacenado en la TSS cuyo selector se asigna en el jmp far. ¿Qué consideraciones deberíamos tener para poder realizar el primer cambio de contexto? ¿Y cuáles cuando no tenemos tareas que ejecutar o se encuentran todas suspendidas?

→ Si es el primer cambio de contexto, es importante que la tarea inicialice el TSS con valores en los registros que permitan la correcta ejecución del código. Cuando no tenemos tareas que ejecutar se hace el salto a la tarea IDLE, que cargara su correspondiente TSS y mantendrá al procesador ocupado. (Al momento de encender la máquina tendremos que hacer un salto manual a la tarea de inicio, que también tendrá su respectivo TSS y saltará a la tarea idle)

4. ¿Qué hace el scheduler de un Sistema Operativo? ¿A qué nos referimos con que usa una política?

→ El scheduler se encarga de decidir cuál es la próxima tarea que el procesador deberá ejecutar, esto lo hace en cada tic del clock, para esto tiene en cuenta una “política” que es una serie de heurísticas que ayudan a decidir cual es la tarea que necesita ser ejecutada primero. (Aparentemente, esta además puede tener en cuenta performance o eficiencia energética, según la configuración del usuario)

5. En un sistema de una única CPU, ¿cómo se hace para que los programas parezcan ejecutarse en simultáneo?

→ Lo que hace es alternar entre múltiples tareas a grandes velocidades, ejecutando pequeñas partes del código y aparentando simultaneidad.

11. Estando definidas sched_task_offset y sched_task_selector:

```
sched_task_offset: dd 0xFFFFFFFF
 sched_task_selector: dw 0xFFFF
```

Y siendo la siguiente una implementación de una interrupción del reloj:

```
global _isr32
 _isr32:
 pushad
 call pic_finish
 call sched_next_task
 str cx
cmp ax, cx
 je .fin
 mov word [sched_task_selector], ax
 jmp far [sched_task_offset]
 .fin:
 popad
 iret
```

a) Expliquen con sus palabras que se estaría ejecutando en cada tic del reloj línea por línea

→ Lo primero que se hace es preservar los registros de propósito general. Luego, se le indica al pic que la interrupción fue atendida. Después se le pide al scheduler la próxima tarea a ejecutar, nos devuelve en ax el selector de segmento de dicha tarea. Para poder comparar la próxima tarea con la actual y así decidir si es necesario intercambiar las tareas, se lee el registro TR(que indica cual es la tarea actual). Se hace la comparación y si son distintas, se mueve el valor de ax a la posición de memoria reservada para el selector y luego salta al selector de TSS en la GDT de la tarea próxima. Por último se obtienen los registros de propósito general y se usa iret para volver a la rutina que la llamó restaurando el eip.

b) En la línea que dice jmp far [sched_task_offset] ¿De que tamaño es el dato que estaría leyendo desde la memoria? ¿Qué indica cada uno de estos valores? ¿Tiene algún efecto el offset elegido?

→ En el jmp far se van a leer 48 bits (6 bytes) de memoria en donde los primeros 32 bits (4 bytes) son de offset en la tss y los últimos 16 son de selector de segmento de la próxima tarea.

c) ¿A dónde regresa la ejecución (eip) de una tarea cuando vuelve a ser puesta en ejecución?

→ Cuando se termina una tarea la EIP va a volver al valor que estaba guardado en su estructura tss

12. Para este Taller la cátedra ha creado un scheduler que devuelve la próxima tarea a ejecutar.

a) En los archivos sched.c y sched.h se encuentran definidos los métodos necesarios para el Scheduler. Expliquen cómo funciona el mismo, es decir, cómo decide cuál es la próxima tarea a ejecutar. Pueden encontrarlo en la función sched_next_task.

→ Empieza a recorrer las tareas de forma circular empezando desde la siguiente a la actual, si encuentra alguna en estado TASK_RUNNABLE, la ejecuta, si vuelve a la actual, sale del ciclo, chequea para esta también si su estado es TASK_RUNNABLE, si lo es la ejecuta y sino ya recorrimos todas las tareas y como ninguna esta ejecutable se ejecuta la tarea Idle.
TERCERA PARTE: TAREAS? QUÉ ES ESO?
14. Como parte de la inicialización del kernel, en kernel.asm se pide agregar una llamada a la función tasks_init de task.c que a su vez llama a create_task. Observe las siguientes líneas:
int8_t task_id = sched_add_task(gdt_id << 3);
tss_tasks[task_id] = tss_create_user_task(task_code_start[tipo]);
gdt[gdt_id] = tss_gdt_entry_for_task(&tss_tasks[task_id]);
a) ¿Qué está haciendo la función tss_gdt_entry_for_task?
→ La función tss_gdt_entry_for_task lo que hace es recibir un puntero y con este construye un descriptor para luego cargarlo a la gdt
b) ¿Por qué motivo se realiza el desplazamiento a izquierda de gdt_id al pasarlo como parámetro de sched_add_task?
→ Se realiza gdt_id << 3 para formar el selector de segmento, el cual va a ser recibido y utilizado por el scheduler y el procesador para hacer el cambio entre tareas

15. Ejecuten las tareas en qemu y observen el código de estas superficialmente.

a) ¿Qué mecanismos usan para comunicarse con el kernel?
→Usan la syscall tasks_syscall_draw(para dibujar la pantalla de la tarea actual) y la estructura compartida ENVIRONMENT(es una estructura global accesible por el kernel y el usuario, que les permite acceder a información del sistema como ver el estado del teclado, saber qué tarea está activa)

b) ¿Por qué creen que no hay uso de variables globales? ¿Qué pasaría si una tarea intentase escribir en su .data con nuestro sistema?

→Si hubiesen variables globales, las tareas podrían escribir en memoria global y posiblemente corromper su propio estado o el de otras tareas. Es por eso que solo pueden escribir su propio stack o áreas compartidas controladas por el kernel.

16. Observen tareas/task_prelude.asm. El código de este archivo se ubica al principio de las tareas.
a. ¿Por qué la tarea termina en un loop infinito?

→Porque en nuestro sistema las tareas no terminan, es decir se siguen ejecutando hasta que salte la interrupción del clock y el scheduler ejecute la siguiente tarea.

CUARTA PARTE: HACER NUESTRA PROPIA TAREA

Ahora programaremos nuestra tarea. La idea es disponer de una tarea que imprima el score (puntaje) de todos los Pongs que se están ejecutando. Para ello utilizaremos la memoria mapeada on demand del taller anterior.
Análisis:
18. Analicen el Makefile provisto. ¿Por qué se definen 2 "tipos" de tareas? ¿Cómo harían para ejecutar una tarea distinta? Cambien la tarea Snake por una tarea PongScoreboard.

→ Se definen dos tipos de tareas para diferenciar comportamientos o funcionalidades, por ejemplo la tarea A es la tarea Pong y la tarea B es la tarde Snake. Para ejecutar una tarea distinta podría o crear otro "tipo" TASKC y asignarle la nueva tarea en el makefile; o cambiar alguna de las ya existentes por la nueva tarea. ejemplo si antes tenía TASKB=taskSnake.tsk lo cambio a TASKB=taskPongScoreboard.tsk
19. Mirando la tarea Pong, ¿En que posición de memoria escribe esta tarea el puntaje que queremos imprimir? ¿Cómo funciona el mecanismo propuesto para compartir datos entre tareas?
→El puntaje se escribe en la direccion virtual SHARED_SCORE_BASE_VADDR, q se define como (PAGE_ON_DEMAND_BASE_VADDR + 0xF00). Todas las tareas acceden a la misma direccion virtual SHARED_SCORE_BASE_VADDR, q esta mapeada a la misma pagina física para todas ellas, cada tarea accede con su task_id a una sección reservada de 8 bytes en la página compartida, donde se escribe el puntaje.

PREG PARCIALES VIEJOS

1. A continuación se tienen las entradas de una TLB correspondientes a una misma tarea que ejecuta en un sistema basado en un procesador x86. Las entradas están en el orden en el que se han ido generando las traducciones.

#	Nro.Pag	Descriptor	Ctrl
1	7C047	0EF00121	ccc
2	7EEF0	1EF01067	ccc
3	EC004	001F0163	ccc
4	EC005	001F7123	ccc
5	46104	1F011005	ccc
6	46109	1F010027	ccc

Para dicha tarea el registro **CR3 = 0x000E4000** y las tablas de página correspondientes a las direcciones en uso se alojan en páginas de memoria física inmediatamente contiguas al **DTP** en el orden en el que se han ido requiriendo. A modo de ejemplo: la tabla de páginas correspondiente a la entrada Nro 1 de la **TLB** se ubica en la página de memoria contigua a la del DTP, la Nro 2 se aloja en la siguiente página física, y así sucesivamente. Se pide:

a) Para las entradas 1 y 2 de la TLB, escribir el contenido de sus correspondientes descriptores en cada nivel de la jerarquía de tablas de traducción a direcciones físicas, considerando que se trata de las dos primeras páginas requeridas al ejecutar la tarea.

https://www.cubawiki.com.ar/images/2/2d/Orga2_2parcial_10-11-22_teorico.pdf

b) Especificar para cada **PDE** que valor deben tener los bits **U/S** y **R/W** para ser consistentes con el contenido de la TLB. Respuestas posibles en cada caso: 0, 1 o X (indistinto).

c) Suponiendo que las seis entradas están siendo utilizadas por una misma tarea y por cada task switch se modifica CR3. ¿Qué ocurre con cada una al momento del task switch? ¿Cual es el tratamiento para aquellas que se han modificado?

Al realizar un task switch y cambiar el CR3 necesitamos flushear (invalidar) la TLB porque esta va a contener traducciones viejas que pueden pisarse con las del esquema de paginacion al que no movimos. De ser requerido las traducciones guardadas (cacheadas) como globales si van a permanecer en la TLB, esto nos permitira por ejemplo mantener las traducciones para el identity mapping del kernel, esto se hace con el cr3 guarda las globales y invalida al resto, de ser requerido podria no hacerlo.

d) ¿Cuál es la función dentro de la TLB de los bits identificados genéricamente como **Ctrl**?

Los bits identificados como CTL son los típicos bits de control para una caché. Esta valid qué va a indicar si la traducción es válida o no y LRU (least recently used) qué es el mecanismo que permite implementar la política de desalojo de la traducción más vieja sin uso (o menos uso) al llenar la TLB.

2. ¿Por qué un sistema que utilice dos o más niveles de privilegio diferentes debe usar una TSS aún cuando la conmutación de tareas es manual?

Un sistema de tareas cuando utiliza dos o más niveles de privilegio diferentes debe usar una TSS aún cuando la conmutación de tareas es manual ya que más allá de que podamos hacer la conmutación de tareas a medida siempre vamos a necesitar la TSS para que el CPU pueda hacer el cambio de privilegio automáticamente cuando esto sea requerido (un buen ejemplo es en las interrupción de excepción porque si cambiamos de nivel vamos a necesitar usar otra pila de nivel cero y preservar la pila “de donde veníamos”)

3. Se tiene un sistema SMP. Cada procesador tiene su propio controlador Cache. Utiliza para mantener coherente los sub sistemas de memoria cache y la DRAM el protocolo MESI

(a) ¿Cuál es el recurso de hardware mediante el cual cada Controlador Cache detecta las transacciones que los demás Cores cursan con la memoria del sistema (DRAM)?. Indicar el nombre del recurso, y a qué líneas del bus se conecta.

El snoop bus es el recurso mediante el cual cada Controlador Cache detecta las transacciones que los demás Cores cursan con la memoria del sistema (DRAM) este en particular no le interesan los datos sino que le interesan mas las direcciones y senales de control para saber todos a todos el estados de los cores.

(b) Explicar si M es un estado preciso o impreciso. Justificar.

El estado M (modified) es preciso ya que en estado M la linea dada va a contener el unico valor actualizado del dato. Aunque la DRAM esté desactualizada, la cache del core que tiene el dato

en M **sabe con certeza** que su copia es la correcta y de trabajar con mesi el resto de los cores la invalidara..

(c) En caso que otro procesador inicie una transferencia de lectura sobre un dato que éste procesador tiene en una línea, cuyo estado es M, se desea saber:

i. ¿Están coherentes las diferentes copias del dato requerido? Justificar.

No estan coherentes. El core que tiene la linea en M posee un valor que difiere de memoria pues aun no se escribio en ella (por eso m)

ii. ¿Cómo se asegura la coherencia de la operación?. Justificar. Detallar el handshake de hardware describiendo las líneas involucradas si corresponde.

iii. ¿Cuál es la política de escritura aplicada por el procesador antes de la transacción analizada y después de la misma?