

Segundo Parcial

Primer Cuatrimestre 2023

Corrigió: Ignacia

Normas generales

- El parcial es INDIVIDUAL
- Puede disponer de la bibliografía de la materia y acceder al repositorio de código del taller de system programming, desarrollado durante la cursada
- Las resoluciones que incluyan código, pueden usar assembly o C. No es necesario que el código compile correctamente, pero debe tener un nivel de detalle adecuado para lo pedido por el ejercicio.
- Numere las hojas entregadas. Complete en la primera hoja la cantidad de hojas entregadas
- Entregue esta hoja junto al examen. La misma no se incluye en el total de hojas entregadas.
- Luego de la entrega habrá una instancia coloquial de defensa del examen

Régimen de Aprobación

- Para aprobar el examen es necesario obtener como mínimo **60 puntos**.
- Para promocionar es condición suficiente y necesaria obtener como mínimo **80 puntos** tanto en este examen como en el primer parcial

NOTA: Lea el enunciado del parcial hasta el final, antes de comenzar a resolverlo.**Enunciado****Ejercicio 1 - (70 puntos)**

En un sistema similar al que implementamos en los talleres del curso (modo protegido con paginación activada), se tienen 5 tareas en ejecución y una sexta que procesa los resultados enviados por las otras. Cualquiera de estas 5 tareas puede en algún momento realizar una cuenta y enviar el resultado de la misma a la sexta tarea para que lo utilice de manera inmediata. Para ello la tarea que realizó la cuenta guardará el resultado de la misma en EAX. A continuación, la tarea que hizo la cuenta le cederá el tiempo de ejecución que le queda a la tarea que va procesar el resultado (lo recibirá en EAX). Tener en cuenta que la tarea que hizo la cuenta no volverá a ser ejecutada hasta que la otra tarea no haya terminado de utilizar el resultado de la operación realizada.

Se desea agregar al sistema una syscall para que la tareas después de realizar la cuenta en cuestión puedan cederle el tiempo de ejecución a la tarea que procesará el resultado.

Se pide:

- a. Definir o modificar las estructuras de sistema necesarias para que dicho servicio pueda ser invocado.
- b. Implementar la syscall que llamarán las tareas.
- c. Dar el pseudo-código de la tarea que procesa resultados (no importa como lo procese).
- d. Mostrar un pseudo-código de la función `sched_next_task` para que funcione de acuerdo a las necesidades de este sistema. Responder: ¿Qué problemas podrían surgir dadas las modificaciones al sistema? ¿Cómo lo solucionarías?

Se recomienda organizar la resolución del ejercicio realizando paso a paso los items mencionados anteriormente y explicar las decisiones que toman.

Detalles de implementación:

- Las 5 tareas originales corren en nivel 3.
- La sexta tarea tendrá nivel de privilegio 0.

Ejercicio 2 - (30 puntos)

Se desea implementar una modificación sobre un kernel como el de los talleres: en el momento de desalojar una página de memoria que fue modificada esta se suele escribir a disco, sin embargo se desea modificar el sistema para que no sea escrita a disco si la pagina fue modificada por una tarea específica.

Se les pide que hagan una función que, dado el CR3 de la tarea mencionada y la dirección física de la página a desalojar, diga si dicha pagina debe ser escrita a disco o no.

La función a implementar es:

```
uint8_t Escribir_a_Disco(int32_t cr3, paddr_t phy);
```

Detalles de implementación:

- Si necesitan, pueden asumir que el sistema tiene segmentación *flat*.
- NO DEBEN modificar las estructuras del kernel para llamar a la función que están creando. Solamente deben programar la función que se pide.

A tener en cuenta para la entrega (para todos los ejercicios):

- Está permitido utilizar las funciones desarrolladas en los talleres.
- Es necesario que se incluya una explicación con sus palabras de la idea general de las soluciones.
- Es necesario escribir todas las asunciones que haga sobre el sistema.
- Es necesaria la entrega de código o pseudocódigo que implemente las soluciones.

Entreg 5 hoja

(y que terminas)

1

2 do parcial

- 1) a) Sabemos que los 6 tareas fueron inicializadas en el scheduler.
O sea, cada una de ellas tiene su TSS y un descriptor correspondiente
en el GDT. Los descriptores de las 6 tareas de nivel 3 como
que de nivel 0 tienen los mismos atributos en el descriptor ($DS=0$, $S=0$)
- Así así, los TSS de las tareas de nivel 0 deben tener los
seletores de segmento de código y datos de nivel 0 ~~DS=0 y CS=0~~ (GDT_CODE=0_SEL
y GDT_CODE=0_SEL). En nuestro sistema del taller no tenemos una función
que crece tareas para que alcancen a nivel 0, por lo que habrá que
que crea esa función (hay que leer en Cweb que se debe pedir una página
del kernel para el stack de la tarea).
 - No hace falta ~~pedir~~ pedir una página del kernel para el stack de nivel
0 (pues ya su propio stack de tarea es de nivel 0) por lo que no se
necesaria en el contexto de rastreo el lanzar la interrupción, más de que
el campo ES0 de su TSS puede tener cualquier valor.
 - También cabe notar que hay una función que inicializa las
estructuras de paginación con páginas de código y datos de nivel 0 (superpage),
pero mmu_init_task_dir ~~funciona~~ crea estructuras de paginación de
nivel 1 con páginas de código y datos de nivel 3.
 - ~~Por tanto~~ Crear una entrada en el IDT para lo sys call. Definir que
su id es 47 (no se coloca con ningún código si deseo la misma
otra interrupción en nuestro sistema)

Datos sobre ~~el sistema~~ & crea lo entrada con lo mío:

IDT_ENTRY 3(47); No

En la linearización de lo IDT. Notar que debe ser una interrupción con DPL=3 para que pueda ser invocada por cualquier tarea a nivel de memoria.

b) La idea de la ISR de lo siguiente:

Le está ejecutando la tarea A, y el scheduler define que la siguiente tarea a ejecutar es la B

2

Si tarea A hace lo syscall, por lo que se debe deshabilitar la ejecución de la tarea A en el scheduler y se debe habilitar la ejecución de la tarea B. También se guarda en la memoria el id de la tarea A, para que luego pueda volver a ser habilitada. Una vez hecho eso se hace un jmp far al TR de la tarea B, para así lo que contenga en cambio de contexto, guardando en el estado de la tarea A en su TSS y restableciendo el contexto de la tarea B. Notar que:

- * Cuando se cambia a la tarea B, pero el scheduler se sigue ejecutando la tarea A porque no se da, esto es algo en lo que cuando ocurre la interrupción de reloj durante la ejecución de la tarea B se cambia a la tarea B (que es la que sigue a A), interrumpiendo el orden de ejecución ejecución de la tarea A.
- * Al cuando se manda a habilitar la tarea A en ejecución continua en el punto de lo irq, luego se restauran los valores de sus registros y se vuelve a la ejecución de su contexto.
- * Cuando se produce la interrupción de reloj se guarda el contexto de la tarea B, por lo que una ejecución continua en ese punto. El Estado de la tarea A lo cambia.
- * Al deshabilitar la tarea A, el scheduler lo selecciona al finalizar la tarea que sigue al hacer sched-next-task, por lo que se selecciona la tarea siguiente.
- * Solamente ocurre lo inverso con la tarea B, pues el otro habilitado visto ejecutado es si es la tarea siguiente a sched-next-task

global _isr47

_isr47:

pushad ; Guarda los registros de registro para el tarea.

push eax ; hace el punto para la función habilitar_tarea

(0000000000000000)

call habilitar_tarea_6

popad (0000000000000000)

pop eax

jmp far [task_G_offset] ; // Llego al Cabello a lo Tarea 6

popad

↓ ¿Cuál es el valor del selector?

iret

↓ E. de 32 bits para eax

uint8_t habilitar_tarea_6 (uint32_t resultado)

// Uso current_task como variable global y task_6_id

Sched_disable_task(current_task);

Sched_enable_task(task_6_id);

// Ahora ocupo la TSS de la tarea 6 para saber que en eax lleva

// el resultado de la tarea A

// Busco el selector ^{TSS} de la tarea 6 en sched_entry_t, que tiene los descriptores de

// todos los tareas.

uint16_t index_t6_tss = sched_entry_t[task_6_id] >> 3; // Ignoro los bits RPL y T

uint16_t selector = (index_t6_tss & 0x0f) << 16

void* tss_t6_addr = (gdt[task_t6_id].base_31_24 & << 24)

| (gdt[task_t6_id].base_23_16 << 16)

| (gdt[task_t6_id].base_15_0);

tss_t6 = (tss_t6_tss) tss_t6_addr;

3

task_6_offset: tss.t6 → cax = resultado;

/* En todo el proceso anterior se accedió al descriptor de TSS (que está en la GDT) y
posteriormente ocurren a la TSS de la tarea 6 y cambia el valor de registro cax. */
tarea_desalojada = current_task; // Es una variable global del scheduler

return current_task;

}

task_6_offset dd 0

task_6_selector dw 0

Esta sería la inicialización que señala la corrección de la página anterior.
Me faltó detallar que task_6_selector es la parte de la memoria que guarda
el TR de la tarea 6, para así poder hacer el cambio de tareas.
No es necesario darle algún valor a task_6_offset,
pues será ignorado al hacer el jmp far.

Al llegar al de selector Esto parte de lo mismo que ver la inicialización
el kernel inicia la tarea 6, dentro de la memoria del kernel para que otra tarea no pueda
intervenir

④ De modo similar al de su función anterior, para ello se va
al restaurar su TSS, la tarea 6 tendrá en cax el resultado de
la tarea A que lo invocó. Esta tarea debe ser inactivada en el estado
de PAUSE por el scheduler para que no se ejecute hasta que sea
requerido

TAREA 6:

while(true) {

 // Realiza sus tareas

 // Procesa el dato

// Una vez procesado se da un interrupción
// y regresa con la siguiente tarea

 // Continúa la ejecución
 // Continúa otras

// Habilito la tarea A

Sched-Enable-task(tarea_desalojado);

↳ sched-disable-task(task_b_id);

Cambiar-tarea();

} // Curro del ciclo while

}

En global Cambiar-tarea

Cambiar-tarea:

pushad

call sched-next-task

loop

de ad

push word [sched-task-selector], ax

loop

mov word [sched-task-selector], ax

jmp far [sched-task-offset]

popad

ret

Liberacion de tarea.

Todo el procedimiento de TAREA 6 se ejecuta en un ciclo que lo tiene
lo ciclo de esto se que, una vez que TAREA 6 terminó su ejecución, se
libera el ciclo deshabilita a su misma (pues al liberar debe terminado lo
que sigue ejecutándose) y habilita la ejecución de la tarea A (que fue la
que libero la syscall) para luego tener el control de este contexto a la tarea

4

que le responde al scheduler.

Una vez hecho eso, si otra tarea B ejecuta la sys call interrupter y el estado de la tarea 6 es el popad de la función Cambiar_tarea, bien el ret y así rebota al principio del ciclo, ~~antes que~~ ^{después} que el scheduler el pausado la tarea para que la tarea 6 pase ^{popa} le que tiene que hacer. ~~el resultado~~ con el resultado de la tarea B.

Notas que:

- De este manera la tarea 6 muere temprano, sin que el finalizar su procesamiento, otros invocados de muerte no reute el control de la tarea.
- La tarea puede hacer uso del scheduler por su id (id 0).

En Cambiar_tarea se hace el cambio muy rápidos o como lo hace la interrupción del reloj, se reutilizan las prioridades de horario de sched_task_selector y offset porque se activa su hidrúfusión en la interrupción del reloj, pero ellos podrían también utilizar otras prioridades de horario.

~~task_reschedule~~

d) Como ha sido necesario implementar las finalizaciones de tareas, no hay bucle que contiene en el sched-next-task.

Un círculo se hace uso de la función sched_disable_task que ha de ser utilizada en los tolling y también se hace uso de sched_enable_task para fuera de la finalización de las tareas.

Se agregan los variables globales:

- task_b_id Cuyo valor será asignado al thread id de la tarea B.
- tarea_desalojada. Muchos no entienden mi razonamiento cuando una tarea hace una sys call. Notas que se guarda en memoria.

Los privilegios que pueden surgir son:

- Que uno tiene A ~~llamó~~ llamó a la syscall ~~que~~ lo ejecución de la tarea 6 no finalizó y otra tarea B llame a la syscall. Dicho modo no hay forma de establecer el resultado de la ejecución porque se pierde el valor de tarea-desajusta (para solucionar esto se podrían implementar ~~los~~ buffers de los resultados generados desde la TSS de otras, en este caso se devolvería el resultado de la tarea 6 en la cola en lo que se presencia en ella el id de la tarea y su resultado). De otro modo, la tarea 6 hace uso de la cola utilizando tanto el id como el resultado del tope de la cola. Al final del ciclo se presenciará y solo tendrá su ejecución cuando la cola esté vacía.
- Que lo propio tarea 6 llame a la syscall (ha hecho)

* Como nota, no hay control de privilegio cuando la tarea 6 lo interrumpe por el reloj, pero no afecta la tarea porque el iret mantiene el privilegio de la tarea interrumpida.

④ Por esto no se quita memoria al resultado desde la TSS de la Tarea 6, sino que el inicio del ciclo comienza con el valor resultado de la web.

5

Hice el ejercicio pensando que la página tenía que ser accedida, no modificada, así que cada vez que dije **accedida** tendría que haber escrito **modificada**. La resolución del ejercicio no cambia mucho por eso.

2

Para hacer esto tengo que borrar todo el directorio de páginas jí
la tarjeta, y de aquella PT que con métodos recorre todos sus directorios
de página para borrar si alguno de ellos tiene como dirección fiance de ghi.
Y todo lo que se guarda se borra porque no tiene otra dirección estable.

1. *Leucosia* (L.) *leucostoma* (L.)

pero luego chegar en el descryptor si que ocurredio.

'La función de trabajo' 1 en tres direcciones: física, mental y social

uint8_t Escribir_a_Disco(uint32_t cr3, void*addr_t phy) {

```

pdUInt32_t pdAddr = (Cr3 & 0xFFFFF000); // ignore last 12 bits when symbulation
pd_entry_t * pd = (pd_entry_t *) pdAddr;
uint32_t res = 0;

```

For(int i=0; i < 1024; i++) {

$$\text{pt_entry_t} * \text{pt} = (\text{pt_entry_t} *) \text{pd}[i],$$

11 he fijo al Re il suo estratto velato (lo uso in modo di far partire in 1)

$i.F((pt \rightarrow attr) \wedge MMU_P) == 1 \}$

$\text{obyes} = \text{chequeo_pt}(\text{pt}^*, \text{phy});$

۳

3

return res

{

Chaque pt tiene cada uno de sus estados de página, si los estados se validan, se fija su dirección física, si coinciden con pte, se fija si fue ejecutado.

```

uint8_t Chequeo_pt(pt_addr_t *pt, paddr_t phy)
{
    uint8_t res = 0;
    for(int j=0; j<1024; j++) {
        if ((pt[j] & attr & MMU_P) == 1) { // Chequeo whether
            if ((pt[j] & page) == e) // if page = e
                if (((pt[j] & page) << 12) == (phy & 0xFFFFF000)) { // Chequeo que en la dirección
                    // busco
                    if ((pt[j] & attr & MMU_A) == 1) { // Chequeo que haya sido accedido
                        res = 1; // Es ese el bit?
                        break; // Si se cumple, no necesito revisar más
                    }
                }
        }
    }
    return res;
}

```

Defino $MMU_A = 1 \ll 5$;

~~Revisa que las direcciones de memoria estén bien formadas~~
~~No queremos que la dirección que lo sitúe sea la dirección que lo sitúe como si fuera una dirección virtual que tiene que ser la dirección física~~
~~Si no es así, no se cumpliría la condición de que la dirección física sea la dirección virtual~~

Notar que tanto la función principal podrán terminar en ignorar el encabezado que la página fue accedida, pero si dentro de eso, que termina en ejecución, nota también que lo que hace dentro de la ejecución que las páginas phy lo fue tratado, pues podríamos haber tratado la ejecución que las páginas phy lo fue tratado, pues podríamos haber tratado dentro de la dirección virtual. Por eso esto puede ocurrir que lo fue accedido al recorrer toda la lista.