

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 haces
terminar)

1

2 do parcial

- 1) a) Sabemos que los 6 tareas fueron inicializadas en el scheduler.
O sea, cada uno de ellos tiene su TSS y un descriptor correspondiente
en el GDT. Los descriptores de las 6 tareas de nivel 3 como
que de nivel 0 tendrán 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 ~~GDT_CODE=0_SEL~~
y ~~GDT_CODE=0_SEL~~). En nuestro sistema del taller no tienen una función
que creen tareas para que crezcan a nivel 0, por lo que habrá que
que crea esa función (hay que leer en cuál 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 verá
necesaria un cambio de nivel al invocar la interrupción, más de que
el campo `ESPC` de su TSS puede tener cualquier valor.
 - También cabe notar que hay una función que iniciaiza la
estructura de paginación con páginas de código y datos de nivel 0 (superpage),
pero `mmu_init_task_dir` ~~crea~~ crea estructuras de paginación de
nivel 1 con páginas de código y datos de nivel 3.
 - ~~Por tanto~~ ~~desde~~ Sabemos entonces que todas las tareas pueden ser inicializadas correctamente.
Pero crear una entrada en el IDT para lo syscall. Definir que
su id es 47 (no se coloca con ningún código si desprecia la página
otra interrupción en nuestro sistema).

Dos procesos ~~funciones~~ crean lo entrada con lo mero:

IDT_ENTRY(47);

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

b) La idea de la ISR es lo siguiente:

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

2

Si la 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 tiene en cache de contexto, guardando en el estado de la tarea A en su TSS y restableciendo el contexto de la tarea B dentro del TSS, por lo que la ejecución continúa en el cache 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 decir que cuando ocurre la interrupción de reloj durante la ejecución de la tarea A se cambia a la tarea B (que es la que sigue a A), interrumpiendo el orden de ejecución ejecución de la tarea A.
- * Si cuando se manda a habilitar la tarea A en ejecución continua en el punto de la ISR47, 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 continuará en ese punto. El estado de la tarea A se cambia.
- * Al deshabilitar la tarea A, el scheduler lo selecciona al hacer la tarea que sigue al hacer sched-next-task, por lo que se selecciona la tarea siguiente.
- * Solamente ocurren la invocación de la tarea B, pues el resto habilitado ya lo ejecutado es si es la tarea siguiente a sched-next-task.

global _isr47

_isr47:

pushad ; Guarda los registros de registro para el bucle.

push eax ; hace el parámetro para la función habilitar_tarea

(0000000000000000)

call habilitar_tarea_6

popad

pop eax

jmp far [task_G_offset] ; // Llamo al código de Task 6

popad

↓ ¿Cuál era el valor del selector?

iret

↓ Es de 32 bits para eax

uint32_t habilitar_tarea_6 (uint32_t resultado)

// Usar 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 operar que en eax sea

// el resultado de la tarea A

// Busco el selector ^{TSS} de la tarea 6 en sched_entry_t, que tiene los descriptors 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 = task_6_selector * 8; tss_t6 → cax = resultado;

/* En todo el proceso anterior se accedi al descriptor de TSS (que está en la GDT) y posteriormente ocurren a la TSS de la tarea 6 y cambia el valor de su 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 selector Esto parte de la memoria que va localizada en el inicio de la tarea 6, dentro de la memoria del kernel para que otra tarea no pueda manipularla

④ De modo similar de la que hicimos anteriormente, 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 inacabada en el estado de PAUSE por el scheduler para que no se ejecute hasta que sea requerido.

TAREA 6.

while(true) {

 // Invocar sus rutinas

 // Procesar el dato

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

 // Continua la ejecución
 // Continua otras

// Habilito la tarea A

Sched-Enable-task(tarea_desalojado);

) Sched-disable-task(task_b_id);

Cambiar-tarea();

} // Cambio del ciclo while

}

En global Cambiar-tarea

Cambiar-tarea:

pushad

call Sched-next-task

loop

ret

push word [Sched-task-selector], ax

loop

mov word [Sched-task-selector], ax

jmp far [Sched-task-offset]

popad

ret

Habilitar la tarea B

Todo el procedimiento de TAREA 6 se ejecuta en un ciclo que lo tiene
los datos de lista de gnu, una vez que TAREA 6 terminó su ejecución, se
llama al sistema deshabilitar a su misma (pues el sistema debe terminado lo
que realizó ejecutándose) y habilita la ejecución de la tarea A (que fue lo
que llamó la syscall) para luego borrar el contexto de TAREA 6 de la tarea

4

que le ríce en el 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, hace el ret y así vuelve al principio del ciclo, ~~antes que el que se ha ejecutado~~ ^{popad} la tarea para que la tarea 6 pase ^{popad} lo que tiene que hacer. ~~después~~ con el resultado de la tarea B.

Notas que:

- De este manera la tarea 6 hace turno, sin que el finalizar sea programado, otros invocados de hacer no reivindica 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 manejo de sched_task_selector y offset porque se activa su hidrúfusión en la interrupción del reloj, pero otras podrían también utilizar otras prioridades de manejo.

~~task_requeue~~

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 llamados y también se hace uso de sched_enable_task para fuera de la finalización de las tareas.

Se agregan las variables globales:

- task_6_id Cuyo valor será asignado al thread id de la tarea 6
- tarea_desalojada. Muchas más se incluirán en otras cuando una tarea haga 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 lo tiene 6 no finalizó y otra tiene B llama a la syscall. El resultado para los hay tienen de restablecer la ejecución de lo tiene 6 porque se pierde el valor de tiene-desajusta (para restaurar esto se ponen los números ~~entre~~ dentro de los ~~entre~~ resultados devueltos desde la syscall de tiene, en tanto se ~~devuelven~~ ~~entre~~ el ~~entre~~ resultado de la syscall Cola en lo que se muestra en ella el id de lo tiene y su resultado ~~que~~ preciso).
- De otro manera, lo tiene 6 hace uso de la cola utilizando tanto el id como el resultado del tope de la cola. Al final del ciclo se ~~desencola~~ y solo tiene su ejecución cuando la cola está vacía.
- Que lo propio tiene 6 llama a la syscall (ha hecho)

* Cabe hacer, no hay cambio de privilegio cuando lo tiene 6 lo interrumpe por el reloj, pero ha efecto la hace porque el iret mantiene el privilegio de lo tiene interrumpido

④ Por esto no se pone nombre al resultado desde la TSS de lo tiene 6, sino que el inicio del ciclo comienza con el valor resultado de lo que

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 recorrer todo el directorio de páginas si se tiene, y de aquella PT que son válidas recorrer todo sus entradas de página para buscar si alguna de ellas tiene como dirección física de phy.
 Notar que de acuerdo a lo visto porque las tres tienen la dirección seteada.
 Porque se ve que

para luego checar en el directorio si fue ocultado.

La función devolverá 1 si esa dirección física fue ocultado y 0 sino

```
uint8_t Escribir_o_Disco(uint32_t cr3, pdaddr_t phy) {
    pduint32_t pdAddr = (cr3 & 0xFFFFFFFF); //Ignora los 12 bits lower significativos
    pd_entry_t * pd = (pd_entry_t *) pdAddr;
    uint32_t res = 0;
    for(int i=0; i < 1024; i++) {
        pt_entry_t * pt = (pt_entry_t *) pd[i];
        // Se fijo si es una entrada válida (lo sé si en el bit present está en 1)
        if((pt->attr & MMU_P) == 1) {
            res |= chequeo_pt(pt, phy);
        }
    }
    return res;
}
```

chequeo_pt hace algo más de las entradas de página, si la entrada es válida, se fija su dirección física, si coincide con phy, se fija si fue ocultado.

```

uint8_t Chequeo_pt(pt_addr_t *pt, phy, paddr_t phy
    uint8_t res = 0
    for(int j=0; j<1024; j++) {
        if ((pt[j] & attr & MMU_P) == 1) { // Chequear whether
            if (pt[j] & page == e) // page = e
                if (((pt[j] & page) << 12) == (phy & 0xFFFFF000)) { // Chequear que en la dirección
                    // busco
                    if ((pt[j] & attr & MMU_A) == 1) { // Chequear 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 las direcciones que lo intentan, pero no sea accedido,~~
~~que no sea que tengan direcciones virtuales que estén mapeadas a la memoria física, lo que obliga a checar tanto los bloques virtuales~~

Notar que tanto la función principal podrán terminar en ignorar el encabezado que la página fue accedido, pero si decidí dejar que terminen en ejecución
 Notar también que lo que hace dentro la ejecución que las páginas phy no fue
 Accedida, pues podrían haber hecho la ejecución sin las páginas phy no fue
 Accedida, pues podrían haber accedido dentro una dirección virtual. Por eso
 esto puede ocurrir que no fue accedido al recorrer todo la entrada.