Segundo Parcial

Primer Cuatrimestre 2021

Normas generales

- El parcial es INDIVIDUAL, de modalidad remota y a libro abierto.
- Las consultas se realizarán por *Zulip*, las mismas serán iniciadas por los docentes a solicitud de los alumnos. Para solicitar consultas, deberán completar su nombre en una planilla compartida y a medida que los docentes podamos ir respondiendo, los contactaremos mediante mensaje privado de *Zulip* en orden de llegada. Las consultas se respondenrán hasta las 22hs.
- Dispondrán de 5hs para resolver el parcial, a las 17hs del día del parcial se publicará un archivo comprimido que deben descargar, el mismo contendrá todo lo necesario para realizar el parcial.
- Una vez cumplido el plazo se deberá completar un formulario con el *hash* del *commit* del repositorio de entrega.
- Se considerará un lapso de 30 minutos (22:00hs a 22:30hs) adicionales para realizar la entrega del parcial, es decir, realizar el commit en el *git*, push al repositorio remoto y el completado del formulario de entrega. Si por algún motivo tienen problemas con *gitlab*, es fundamental que obtengan el hash del commit en su repositorio local y lo envien en el form de entrega. Esto certifica que realizaron la entrega en tiempo y forma. No será corregido ningún parcial cuyo hash no haya sido enviado, o el hash no corresponda con un commit de su repositorio.
- Durante el parcial usaremos el canal de Anuncios en *Zulip* para darles información actualizada y aclaraciones sobre el examen. Les pedimos que lo revisen con regularidad.

Formato de entrega

Se aceptará como solución archivos en formato `TXT` plano, `Markdown` o `PDF`. No se aceptarán archivos `tex` sin compilar o archivos en formatos `doc` o `odt` sin pasar a formato `PDF`. Es válido entregar todo tipo de documentación adicional como imagenes en `jpg` o `png`. El material de la entrega deberá ser separado por ejercicios en carpetas independientes ya creadas en el *template* de la cátedra.

Criterio de aprobación

Cada ejercicio indica una cantidad de créditos. Para aprobar el parcial deberán sumar al menos 65 créditos y haber realizado el ejercicio 2 de forma correcta.

Ejercicio 1 - 10 créditos

Construir un conjunto de segmentos y un mapa de paginación (un solo directorio y varias tablas de página), tal que las siguientes traducciones sean válidas:

Lógica	Lineal	Física	Acción
0x0060:0x00123001	0x00123011	0x01123001	Leer código
0x0060:0x88A94100	0x88A94110	0x00000110	Ejecutar código
0x0030:0x00000000	0xF0000000	0x00000000	Leer Datos
0x0030:0x00399FFF	0xF0399FFF	0x00000FFF	Escribir Datos

Si no es posible completar alguna traducción, justificar, dejando clara cuál es la razón por la cual no es posible realizar dicha traducción. Por simplicidad considerar que todas las traducciones corresponden a acciones realizadas en nivel cero de exactamente 4 bytes

Ejercicio 2 - 50 créditos - Llamen a Moe

Considerar un sistema en modo protegido con segmentación *flat* y paginación activa. Este ejecuta concurrentemente tres tareas de nivel 3 denominadas `Moe`, `Larry` y `Shemp`. Las tareas requieren tan solo 1KB de memoria entre código y datos. Dado el poco espacio que ocupan, estas no resultan muy astutas y suelen realizar accesos a memoria incorrectos. Estos accesos incorrectos deben ser resueltos sin generar ningún error, cualquiera fuere el lugar donde buscan acceder. Como las tareas deben continuar ejecutando a pesar de generar un *page fault*, no se puede continuar desde el mismo lugar y se debe saltear la instrucción que genero el error en su código. A fin de simplificar el mecanismo para saltar una instrucción, las tareas tienen definido el símbolo `ouch` en el offset `0x51` dentro del código de la tarea, desde donde debe continuar la ejecución en el caso generar una excepción de *page fault*. El código en `ouch` requiere que en el estado de los registros sea exactamente el mismo previo a generar la excepción, y que el valor del `EIP` que produjo el error se encuentre cargado en el tope de la pila.

En este simplificado sistema, el scheduler se encarga de ejecutar una a una las tareas intercambiandolas por cada interrupción del reloj. Cada tarea además, tiene asignada un área de pila de 4MB (tener en cuenta que estas tareas no son muy astutas y no saben usar bien la pila).

Por último, las tareas pueden solicitar al sistema la cantidad de veces que este los salvo de ejecutar incorrectamente un acceso. Este valor debe ser un número entero sin signo de 32bits que jamas es reiniciado.

Se pide:

1. Describir cómo se ubicarían en memoria las tareas. Enumerar para ello todos los rangos de memoria física que ocuparía cada parte de cada tarea, y del sistema. Además explicar como se realiza el mapeo sobre direcciones virtuales, considerando que todas las tareas están compiladas para ejecutar desde la dirección virtual `0xCC000000`. Indicar la dirección virtual donde comienza la pila de nivel 3. Detallar donde se ubicarían las páginas que correspondientes a los Page Directory, Page Table y Pilas de nivel cero.

2. Implementar en ASM/C la rutina de atención de interrupciones del reloj. Explicar su funcionamiento.

3. Implementar en ASM/C las rutinas de atención de interrupciones de *page faul¨t*. Explicar detalladamente el funcionamiento de la implementación propuesta. Recordar indicar como es el mecanismo para restaurar los registros pedidos y cargar el `EIP` en la pila.

4. Diseñar e implementar en ASM/C, funcionamiento del servicio para obtener la cantidad de veces que se salvo a la tarea (genero un *page fault*). Indicar en que registro se retorna el resultado.

5. Recordar que las tareas de modo usuario tienen control sobre los registros de propósito general. Un mal diseño de la funcionalidad pedida podría ocasionar un problema de seguridad que le permita a una tarea escribir en una posición arbitraria de memoria. Explicar brevemente cuál sería este problema y cómo se podría remediar.

Nota: Indicar cualquier dato, estructura, variable o función auxiliar referente al sistema que sea necesaria para lograr la implementación propuesta. Por ejemplo: La posición de los descriptores de TSS dentro de la GDT. En caso de requerir algún valor que no se encuentre definido de forma explícita por el enunciado, proponer un valor que considere razonable.

Ejercicio 3 - 40 créditos - Larry esta en cualquiera

Considerar un sistema con segmentación y paginación activa, que ejecuta *n* tareas utilizando dos niveles de protección. Las tareas cada vez que son desalojadas quedan en un estado tal, que al volver a ejecutar, pueden ejecutar las instrucciones `popad` y `iret` para volver a nivel 3.
Se desea implementar las siguientes funciones que serán ejecutadas en nivel 0:

`int getGdtIndex(tss* task)`
La función debe retornar el indice en la GDT del descriptor de la tss pasada por parámetro. En el caso de no existir, la función debe retornar -1.

`int isRunning(tss* task)`
La función debe retornar un 0 si la tarea no se encuentra corriendo, y 1 si la misma se encuentra siendo ejecutada por el procesador.

`void lastThreeReturnPointers(tss* task, void** pointers)`
La función debe buscar en la pila de nivel 3 y retornar cuales fueron las últimas tres direcciones de retorno de funciones por las cuales paso la tarea. El resultado será cargado en `pointers`, el cual es un vector que tiene espacio para tres punteros. Suponer que en todos los casos existen al menos tres llamados a funciones validos para leer en la pila. Además suponer que todas las funciones de la tarea respetan convención C y construyen un stackframe.

Se pide:

1. Implementar en ASM/C la función `int getGdtIndex(tss* task)`.

2. Implementar en ASM/C la función `int isRunning(tss* task)`, puede asumir que la tarea existe en la GDT.

3. Implementar en ASM/C la función `void lastThreeReturnPointers(tss* task, void** pointers)`, puede asumir que la tarea existe y que NO esta siendo ejecutada.

Nota: No se puede asumir tener ninguna estructura adicional para resolver el ejercicio. Las funciones deben ser resultas usando solo las instrucciones que proporciona el procesador. Ej: No se puede suponer que se tiene una tabla con punteros desde a todas las pilas de nivel 3 de todas las tareas.

Ejercicio 1 (10 créditos)

Consigna

Construir un conjunto de segmentos y un mapa de paginación (un solo directorio y varias tablas de página), tal que las siguientes traducciones sean válidas:

Lógica	Lineal	Física	Acción
0x0060:0x00123001	0x00123011	0x01123001	Leer código
0x0060:0x88A94100	0x88A94110	0x00000110	Ejecutar código
0x0030:0x00000000	0xF0000000	0x00000000	Leer Datos
0x0030:0x00399FFF	0xF0399FFF	0x00000FFF	Escribir Datos

Si no es posible completar alguna traducción, justificar, dejando clara cuál es la razón por la cual no es posible realizar dicha traducción. Por simplicidad considerar que todas las traducciones corresponden a acciones realizadas en nivel cero de exactamente 4 bytes

Resolución

Voy a resolver segmentación y posteriormente paginación.

Segmentación

Primero voy a descomponer las direcciones lógicas. Las direcciones lógicas tienen el formato `segsel:offset` y el formato de los selectores de segmento es el siguiente:

```
  SEGMENT SELECTOR
0000000000000 0  00
INDEX         TI RPL
```

Descomposición de direcciones lógicas

Lógica	Selector de segmento					Offset
	Hex	Binario	Índice	TI	RPL	
0x0060:0x00123001	0x0060	0000000001100 0 00	0x0C	0 (GDT)	0x0	0x00123001
0x0060:0x88A94100	0x0060	0000000001100 0 00	0x0C	0 (GDT)	0x0	0x88A94100
0x0030:0x00000000	0x0030	0000000000110 0 00	0x06	0 (GDT)	0x0	0x00000000
0x0030:0x00399FFF	0x0030	0000000000110 0 00	0x06	0 (GDT)	0x0	0x00399FFF

Necesito 2 entradas en la GDT (0x06 y 0x0C).
Sé que al resolver segmentación tengo que obtener la dirección lineal, entonces puedo restar la dirección lineal con el offset de la lógica para obtener la base de cada segmento. Y como tengo dos pares de traducciones con el mismo segmento cada par debería coincidir en la base.

Cálculo de base de segmentos

Índice del segmento	Lineal	Offset lógica	Base (lineal - offset)
0x0C	0x00123011	0x00123001	0x10
0x0C	0x88A94110	0x88A94100	0x10
0x06	0xF0000000	0x00000000	0xF0000000
0x06	0xF0399FFF	0x00399FFF	0xF0000000

Las dos pares de traducciones coinciden en la base!

GDT

Ahora puedo armar un conjunto de descriptores de segmento que cumpla con lo pedido:

GDT

Índice	Base	Límite	G	DPL	Tipo	S	D/B	L	P
0x06	0xF0000000	0xFFFFF	1	0x0	0xA (Code, execute-read)	1	1	0	1
0x0C	0x10	0xFFFFF	1	0x0	0x2 (Data, read-write)	1	1	0	1

Más en detalle:

- Base: se calculó antes `lineal - offset logica`
- Límite/G: El enunciado no da restricciones para el límite así que utilizo G=1 y el máximo de límite (0xFFFFF, 20bits). Esto permite acciones que midan 4 bytes.
- D/B: 1 Default op size. Big 32 bits
- L: 0 (siempre manejamos 32bits)
- AVL: 0
- Present: 1 el segmento tiene que estar presente
- S: 1 (todas son código/datos)
- DPL: 0x0, las acciones se realizan en nivel 0.
- Tipo:
    - Para 0x06: El ejercicio pide ejecutar y leer código en este segmento. Entonces uso 0xA.
    - Para 0x0C: El ejercicio pide leer y escribir datos en este segmento. Entonces uso 0x2.

Paginación

Ahora para resolver paginación descompongo las direcciones lineales que tienen el siguiente formato:

```
  DIRECCIÓN LINEAL
0000000000 0000000000  000000000000
DIR INDEX  TABLE INDEX OFFSET
```

Descomposición de direcciones lineales

Dirección lineal	Binario	Directory index	Table index	Offset
0x00123011	0000000000 0100100011 000000010001	0x000	0x123	0x011
0x88A94110	1000100010 1010010100 000100010000	0x222	0x294	0x110
0xF0000000	1111000000 0000000000 000000000000	0x3C0	0x000	0x000
0xF0399FFF	1111000000 1110011001 111111111111	0x3C0	0x399	0xFFF

- Puedo ver que el offset de la primera traducción es 0x11 y a la dirección física que me piden que la mapee es 0x01123001, lo que es incompatible. Cuando intente buscar la dirección física base para la página (que sólo son los 20 bits más significativos de la dirección física) me quedaría 0x01123 y cuando se resuelva paginación, sumando el offset 0x11 quedaría en 0x01123011 y no en 0x01123001 como pide el ejercicio. Descarto esta traducción.
  Sabiendo que esta traducción no vale, podría cambiar el tipo del selector de segmento que le corresponde de execute-read a sólo execute pero funciona igual.
- Puedo ver que el offset de la cuarta traducción es 0xFFF, y que si intenta realizar una acción de 4 bytes en esta dirección no me alcanza la página, tengo que mapear la página siguiente para permitir acceder a los 3 bytes siguientes.

Directorio de páginas

Ahora armo el directorio de páginas que va a cumplir simultáneamente todas las traducciones:

Directorio de páginas

Índice	Page table entry	U/S	R/W	P
0x222	&PT1 >> 12	0	0	1
0x3C0	&PT2 >> 12	0	1	1

Más en detalle:

- Page table entry: son los 20 bits más significativos de la dirección de memoria donde se encuentra la tabla de páginas correspondiente

- U/S: en este caso utilizamos 0 ya que todas las páginas que se utilizan en el enunciado están en modo supervisor (CPL = 0)
- R/W: lo dejo en 0 para la de código y 1 para el de datos ya que piden lectura y escritura
- P: necesario que esté presente

Tablas de páginas

Y las tablas de páginas correspondientes:

Tabla de páginas (PT1)

Índice	Dirección física base	U/S	R/W	P
0x294	0x00000	0	0	1

Tabla de páginas (PT2)

Índice	Dirección física base	U/S	R/W	P
0x000	0x00000	0	1	1
0x399	0x00000	0	1	1
0x39A	0x00000	0	1	1

Más en detalle:

- Dirección física base: los 20 bits más significativos de dirección física que tiene que mapear cada página. En este caso la calculo shifteando 3 hacia la derecha las direcciones físicas que me pide que mapee el enunciado.
- U/S y R/W: por lo mismo que en el directorio de páginas
- P: necesario que esté presente

Ejercicio 2 (50 créditos) - Llamen a Moe

Resolución

Mapeos

Al comienzo de la memoria estará el Kernel mapeado con identity mapping. A partir de los 4MB se encontrarán las tareas de Moe, Larry y Shemp una tras otra ocupando una página de 4KB (de los cuales los últimos 3KB no se utilizarán).
Luego, a partir de los 8MB se encontrarán las pilas de nivel 3 para cada una de las tareas, una atrás de otra.

Mapeos de memoria

Rango virtual	Rango físico	Información
0 a 1MB	0 a 1MB	Kernel, rutinas de interrupcion y reloj
1MB a 4MB	1MB a 4MB	Área libre del Kernel que guarda las estructuras PD, PT, TSS, y las pilas de nivel 0
0xCC000000 a 0xCC000FFF	4MB+0KB a 4MB+4KB	Código/datos de Moe
0xCC000000 a 0xCC000FFF	4MB+4KB a 4MB+8KB	Código/datos de Larry
0xCC000000 a 0xCC000FFF	4MB+8KB a 4MB+12KB	Código/datos de Shemp
0xA0000000 a 0xA03FFFFF	8MB+0MB a 8MB+4MB	Pila de nivel 3 de Moe
0xA0000000 a 0xA03FFFFF	8MB+4MB a 8MB+8MB	Pila de nivel 3 de Larry
0xA0000000 a 0xA03FFFFF	8MB+8MB a 8MB+12MB	Pila de nivel 3 de Shemp

Por conveniencia, las direcciones virtuales del código y la pila de cada tarea coinciden.
Las pilas de nivel 3 para todas las tareas comienzan en la dirección 0xA0400000.
Los Page Directory, Page Table y Pilas de nivel cero se ubicarán en el área libre del Kernel, entre 1MB y 4MB físicos.

GDT

Voy a necesitar las tareas definidas en la GDT, voy a poner una atrás de otra a partir de 0x5.

GDT

Índice	Base	Límite	DPL	Tipo
0x00	NULL			
0x01 - 0x4	Código/Datos de nivel 0 y 3			
0x5	&tss_moe	sizeof(tss_t)-1	0x0	0x9 (Task 32bit)
0x6	&tss_larry	sizeof(tss_t)-1	0x0	0x9 (Task 32bit)
0x7	&tss_shemp	sizeof(tss_t)-1	0x0	0x9 (Task 32bit)

IDT

Voy a utilizar las siguientes entradas en la IDT:

IDT Entries

Índice	Code segment selector	Offset	DPL	Type	P
14 (page fault)	CODE_SEL_RING_0 << 3	&_isrFault	0x0	Interrupt gate	1
32 (clock)	CODE_SEL_RING_0 << 3	&_isrClock	0x0	Interrupt gate	1

IDT Entries

Índice	Code segment selector	Offset	DPL	Type	P
88 (cantSalvadas)	CODE_SEL_RING_0 << 3	&_isr88	0x3	Interrupt gate	1

`cantSalvadas` tiene DPL 0x3 por que quiero que una tarea nivel 3 (Moe, Larry y Shemp) la puedan llamar con int 88 si quieren.

Aclaración

Voy a definir variables que se van a usar en distintos ejercicios, por ejemplo la variable `current_task` la defino una vez y la utilizo en ejercicios siguientes.

Rutina reloj

Cada ciclo de reloj se va a intercambiar la tarea que está ejecutando por la siguiente tarea en round-robin.

```
task_offset:     dd 0x00000000
task_selector:   dw 0x0000

extern sched_next_task

_isr32: ; handler reloj
    pushad ; se guardan los registros que estaban antes de la interrupcion
    call pic_finish1 ; se avisa al pic que ya recibimos la interrupcion

    call sched_next_task ; busco el selector de segmento de la próxima tarea a ejecutar
    str cx ; leo el selector de la tarea que se está ejecutando
    cmp ax, cx ; comparo el próximo con el actual, no debo saltar si coinciden
    je .fin
        ; salto
        mov [task_selector], ax
        jmp far [task_offset]
    .fin:

    popad ; restauro los registros
    iret
```

```
int current_task = 0;

// devuelve el selector a la siguiente tarea a ejecutar
// round-robin
uint16_t sched_next_task() {
    current_task++;
    current_task %= 3;
    return (0x5 + current_task) << 3;
}
```

Rutina de atención de interrupción page-fault

Yo sé que la pila de nivel 0 tiene esta pinta tras la excepción:

```
|     ...    |
|     SS     |
|     ESP    | = esp+16
|   EFLAGS   |
|     CS     |
|     EIP    | = esp+4
| Error Code | ← esp (la excepción 14 tiene EC)
|     ...    |
```

Con esto sé donde tengo que ir a buscar el EIP anterior y la pila de nivel 3.
Funcionamiento de la excepción:

- Guardo todos los registros el momento de la excepción con `pushad`

- Leo los valores en el stack de nivel 0: el EIP anterior y la pila de nivel 3. **Con cuidado por que ahora el `pushad` pusheó 8 registros al stack de nivel 0 entonces hay que sumar 32 a los offsets.**
- Guardo en la pila de nivel 3 el valor de EIP e incremento el ESP de nivel 3 (que esta guardado durante la excepción el stack de nivel 0)
- Aviso al sistema que hubo un page fault para incrementar el contador (`fault_task`)
- Restauro todos los registros previo a la excepción (como pide el ejercicio)
- Reemplazo en el stack de nivel 0 el EIP de retorno, le hardcodeo `0xCC000000 + 0x51` que es donde se encuentra en memoria virtual el símbolo `ouch`.
- Ahora `iret` va a retornar a `ouch` y la pila va a tener a EIP en el tope.

```
extern fault_task

_isrFault: ; handler page-fault
    pushad ; guardo el estado de los registros al momento de la excepción

    ; guardo EIP en la pila de nivel 3
    mov ebx, [esp + 36] ; leo el EIP anterior
    mov eax, [esp + 48] ; leo el ESP de nivel 3
    mov [eax -  4], ebx ; pusheo EIP en el stack de nivel 3
    sub [esp + 48], 4   ; (parte del push, agrandar el stack en 1)

    ; registro que hubo un page fault en esta tarea
    call fault_task

    popad ; devuelvo los registros a su estado previo a la excepcion (pero con esp-4 ya que guardé EI

    ; cambio la dirección de retorno por ouch
    mov [esp + 4], 0xCC000000 + 0x51
    iret ; va a saltar a ouch
```

```
uint32_t num_faults[3] = { 0 };

void fault_task() {
    num_faults[current_task]++;
}
```

Servicio cantSalvadas

El resultado se devuelve por `EAX`.

```
_isr88: ; handler cantSalvadas
    sub esp, 4 ; reservar espacio para el retorno
    pushad

    call cant_salvadas
    mov [esp + 32], eax ; guardo el retorno

    popad
    pop eax ; devuelvo el resultado en EAX
    iret
```

```
uint32_t cant_salvadas() {
    return num_faults[current_task];
}
```

Vulnerabilidad

Hay una potencial vulnerabilidad en la implementación. La tarea tiene control total sobre el valor de `esp`. Si no se verifica que la pila esté en el rango correcto (es decir, `esp` entre 0xA0000000 y 0xA0400000) podría malintencionadamente escribir a una posición de memoria arbitraria, por ejemplo, direcciones dentro del Kernel. Esto puedo hacerlo porque estoy desde la interrupcion corriendo en nivel 0.
Las siguientes dos líneas extraídas de mi implementación sería donde ocurriría la escritura a memoria arbitraria:

```
mov eax, [esp + 48] ; leo el ESP de nivel 3 (eax es arbitrario!!!1!)
mov [eax -  4], ebx ; pusheo EIP en el stack de nivel 3 (escribo en eax, que puede ser arbitrario!!!1
```

Para remediar el error habría que verificar que el valor de `esp` esté en el rango que debería estar.
Este es el código que debería agregar a mi implementación justo después de guardar `esp` en `eax`:

```
cmp eax, 0xA0000000
jl .vuln ; salta si es menor al minimo
cmp eax, 0xA0400000
jg .vuln ; salta si es mayor al maximo

...

.vuln:
    ; no está determinado que hacer cuando una tarea intenta cometer este delito
    ; podría ser saltar a la siguiente y reiniciar esta
    ; como no está definido lo dejo a definir
    ; ... TBD ...
    iret
```

Ejercicio 3 (40 créditos) - Larry esta en cualquiera

Resolución

getGdtIndex

```
global gdt_desc

gdt_desc: dd 0 0 ; mide 8 bytes

cargar_gdt: ; carga la gdt
    sgdt gdt_desc
    ret
```

```
extern gdt_desc;

int getGdtIndex(tss* task) {
    cargar_gdt();
    gdt_entry* entry = (gdt_entry*) gdt_desc->gdt_base;
    gdt_entry* limit = (gdt_entry*) gdt_desc->gdt_base + gdt_desc->gdt_length;

    int i = 0;
    while(entry < limit) {
        if(entry->p == 1 && entry->s == 1) { // está presente y es de sistema
            if(entry->type == 0x9 || entry->type == 0xB) { // es TSS
                tss_t* tss_gdt =
                    (entry->base_31_24 << 24) |
                    (entry->base_23_16 << 16) |
                     entry->base_15_0;
                if(tss_gdt == task) { // la tss en la gdt coincide con el parámetro
                    return i;
                }
            }
        }
        i++;
        entry++;
    }

    return -1;
}
```

isRunning

```
int isRunning(tss* task) {
    cargar_gdt();
    gdt_entry* base = (gdt_entry*) gdt_desc->gdt_base;

    return base[getGdtIndex(task)]->type == 0xB; // 0x9=TSS, 0xB=TSS Busy
}
```

lastThreeReturnPointers

```
void lastThreeReturnPointers(tss* task, void** pointers) {
    // stack de nivel 3
    // sé que [ebp+4] tiene la dirección de retorno de la última función
    utin32_t* base_stack = task->ebp;

    for(int i = 0; i < 3; i++) {
        // pointers[i] = [ebp+4]
        pointers[i] = *(base_stack + 1); // +1 por que es utin32_t*
```

```
        // avanzo al siguiente stack frame
        // base_stack = [ebp]
        base_stack = *base_stack;
    }
}
```

Correcciones

ej1: B (9)
Segmentación: B
Paginación:
- En la entrada de PT2 con índice 0x39A, la dirección física de la base debe ser 0x00001, si no estamos mappeando dos veces la página 0 (y no vamos a poder acceder a los tres últimos bits de la escritura de la cuarta traducción).

ej2: B (48)
2.1) B
2.2) B muy bien explicado
2.3) B explicado detalladamente el funcionamiento de la implementación, resuelve correctamente.
- Falta remover el error code del stack antes de hacer el iret.
2.4) B
2.5) B

ej3: B (35)
3.1) B
- Falta chequear limite de las entradas de la GDT (que sea mayor a 0x67)
3.2) B
3.3) B-
- Faltaría cambiar el cr3 para que se resuelvan correctamente las direcciones de la pila (ebps) de la tarea