25	38	32,5	95,5 (A+)

Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma no se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

Ej. 1. (25 puntos)

(25p) a. Considerando la siguiente tabla de traducciones de direcciones por segmentación y paginación. De ser posible, dar un conjunto de descriptores de segmento, directorio de paginas y tablas de paginas que cumplan con todas las traducciones **simultáneamente**. Detallar los campos de todas las estructuras involucrados. Además indicar desde que segmento de código se esta ejecutando cada acceso, si la traducción es *identity mapping* y en el caso que alguna traducción no sea posible indicar, ¿por qué?

Lógica	Lineal	Física	Características
0x0193:0x836401A7	0x960C31A7	0x332A21A7	Lectura de 4 bytes, como nivel 0 a nivel 3
0x01A0:0x0392412A	0x960C312A	0x332A212A	Lectura de 2 bytes, como nivel 0 a nivel 0, solo lectura
0x02A8:0x834AAFFF	0x8397AFFF	0x8A9FEFFF	Escritura de 4 bytes, como nivel 0 a nivel 0
0x02B2:0x01B10FFF	0x01B10FFF	0x0010FFFF	Escritura de 1 byte, como nivel 2 a nivel 2

Ej. 2. (40 puntos)

Sea un sistema con segmentación flat y paginación activa, en dos niveles de protección, que ... concurrentemente un conjunto de hasta 1024 tareas independientes. Estas tienen acceso al servicio `NewTask`, que crea una instancia de tarea de entre tres posibles códigos almacenados en el sistema. El servicio toma como parámetro en el registro `eax` un número del 1 al 3 que indica cuál es el código a cargar en la nueva tarea. Además, en este mismo registro se debe retornar uno de los siguientes valores:

- 1 a 3: La nueva instacia de tarea fue creada, indicando de qué tipo fue.
- -1: Ninguna tarea nueva fue creada, indicando que un error.

Considerar que los registros restantes no deberán ser alterados por la ejecución del servicio.

En el caso de la nueva tarea, esta debera almacenar en `edx` el `id` de la tarea que la creo.

(10p) a. Definir un posible mapa de memoria. Indicar el rango de direcciones de:

- paginas de kernel.
- paginas de código y datos de las tareas. Indicar su tamaño.
- paginas donde se mapean las tareas creadas

Además, indicar que información específica es almacenada en cada rango designado. Por ejemplo: *Page Directory*. Explicar el esquema de paginación utilizado detallando el mapeo de cada área de memoria junto con sus atributos. Preferentemente realizar un dibujo del mismo.

nos decimos donde esta todo

(5p) b. Definir las entradas en la IDT para el servicio `NewTask`, para las rutinas de excepciones y para la interrupción de reloj. Complete todos los campos necesarios.

(25p) c. Implementar en ASM y/o C la rutina del servicio `NewTask`

Se cuenta con las siguientes funciones:

- `tss* tss_getFree()`: Retorna una `tss` libre para una nueva tarea. Si no existe `tss` libre retornará *null*
- `uint32_t sched_getId()`: Retorna el `id` de la tarea actual.
- `void sched_add(tss*)`: Agregar un puntero a una `tss` dentro del scheduler para ser ejecutada.
- `pde* mmu_newPD()`: Retorna un directorio de paginas donde todas sus entradas son no presente
- `void mmu_mapPage(pde* cr3, void* virtual, void* fisica, uint8_t us, uint8_t rw)`: Mapea la pagina `virtual` al marco de pagina `fisica` en el mapa de memoria dado por `cr3` con los atributos `us` y `rw`

Ej. 3. (35 puntos)

Se desea implementar una funcionalidad de kernel que cada vez que se desaloje una tarea dentro de la rutina de atención de interrupciones del reloj, se almacene en que función del código de la tarea se produjo la interrupción.

(10p) a. Suponiendo que se cuenta con una función que indica la próxima tarea a ejecutar. Construir una posible rutina de atención de interrupciones de reloj que utilice dicha función para intercambiar tareas. Explicar el funcionamiento de la rutina y de la función que indica la próxima tarea. ¿Qué pasa en el caso que el intercambio de tareas sea por la misma tarea?

(25p) b. Modificar la rutina anterior para agregar la funcionalidad pedida. Considerar que se cuenta con la función `void logEIP(uint32_t gdtIndex, void* func)`, que toma el índice en la GDT de la tarea que se encontraba ejecutando y el puntero a la función que se estaba ejecutando.

Nota: Asumir que todas las subrutinas dentro del código fueron llamadas mediante la instrucción `call`. Esta ocupa exactamente 6 bytes, siendo los ultimos 4 bytes la direccion de la función a llamar

Ejercicio 1

SELECTOR DE SEGMENTO Y OFFSETS EN DIRECTORIO Y PAGE TABLE

TI ↑

caso 1 0x0193 0000 0001 1001 0011 seg código

50 = 32 + 16 + 2 ↓ RPL = 3 lvl 3

0x960C3 1001 0110 0000 EPL = máx (RPL, CPL)

2 5 8 , 0C3 = máx (3, 0) = 3

offset dir offset pt

caso 2 0x01A0 0000 0001 1010 0000 seg CODE LVL 0

52 = 32 + 16 + 4 RPL = 0

0x960C3 258 , 0C3 EPL = máx (0, 0) = 0

offset dir offset pt

Obs.: En ambos casos 1 y 2 se quiere leer en la misma page table con EPL = 3 y EPL = 0 respectivamente una misma página que debe permitir acceso nivel 3 y 0 a la vez. Pero en este último caso se pide que sea "read only". Para ello debería tener en CR0.wp prendido el bit para proteger la escritura por parte del nivel 0, ya que este de lo contrario puede leer o escribir sin importar el atributo R_w de la página. Ahora bien si lo activo, en el caso N°3 tendría problemas pues es nivel 0 con lectura y escritura permitida (no aclara "solo lectura como caso N°2). Por lo tanto este caso N°2 no se puede hacer en simultáneo con el resto.

caso 3 0x02A8 0000 0010 1010 1000 SEG_CODE-LVL-0

85 = 64 + 16 + 5 RPL=0 EPL = (0, 0) = 0

0x8347A 1000 0011 1001

2 0 E , 17A

offset dir offset PT

caso 4 0x02B2 0000 0010 1011 0010 EPL = (2,2) = 2

86 = 64 + 16 + 6 RPL = 2

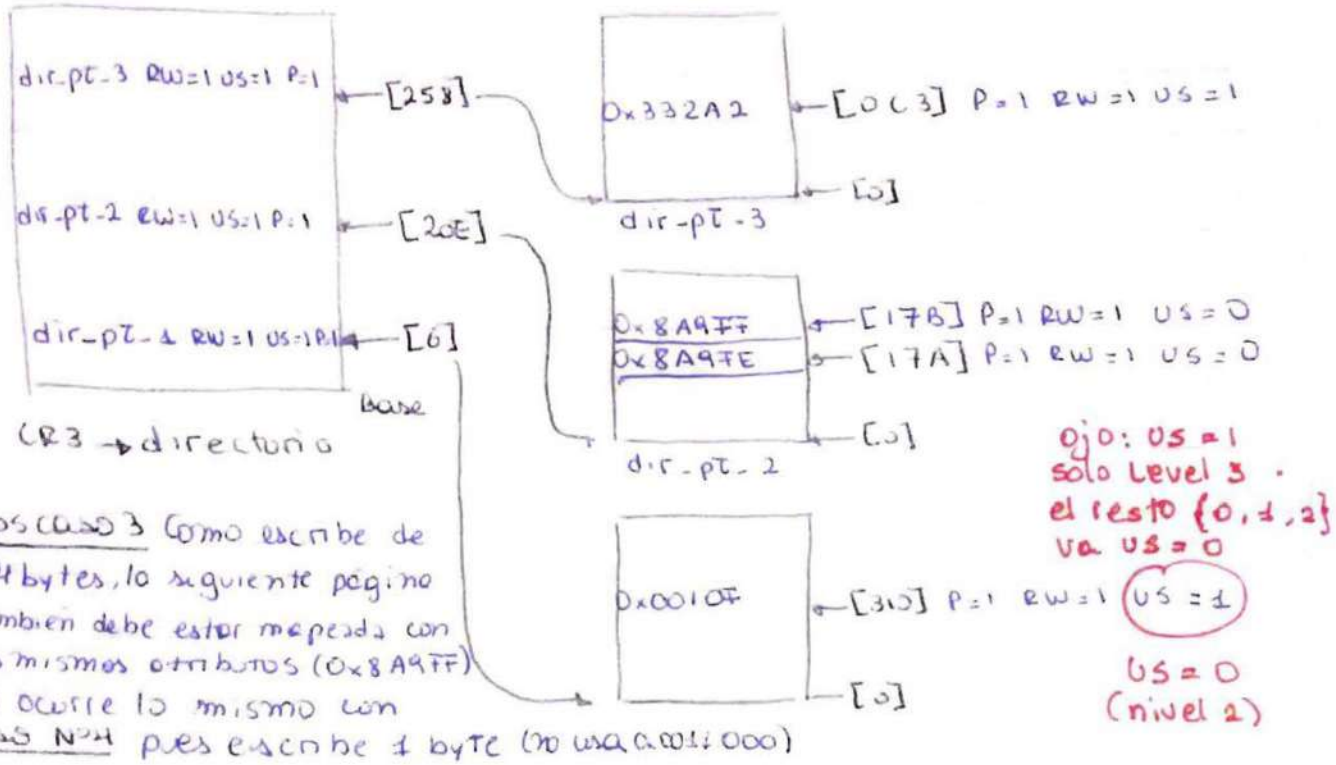
SEL_SEG_LVL 2
CODE

0x01B10 0000 0001 1011

0 0 6 , 3 1 0

off dir off pt

Directorio las entradas no declaradas tienen P=0.



Obs caso 3 Como escribe de a 4 bytes, lo siguiente página también debe estar mapeada con los mismos atributos (0x8A9FF)
NO ocurre lo mismo con caso N°4 pues escribe 1 byte (no usa 0x001:000)

Recordemos que dir_base_seg + offset = dir_lineal

caso 1	Base	0x12A830A0	caso 3	0x00HD0000
+	offset	0x8364 01A7		0x834AFFFF
=	lineal	0x960C31A7		0x8397FFFF

Elijo limit 0x88000 pues 0x88000FFF > offset

caso 4 offset = lineal - base = 0x0000000000

caso 3 → sería lo mismo pero como no se puede 0x02000FFF > offset

Entonces la GDT queda

INDICE	BASE	LIMIT	G	P	DPL	TIPO
50	0x12A830A0	0x88000	1	1	3	read/write not
85	0x00HD0000	0x88000	1	1	0	read/write not
86	0x00000000	0x02000	1	1	2	read/write not
			=			

Ejercicio 2

(a) Necesito que c/ tarea pueda acceder a las 3 páginas que tienen los códigos posibles. Como son independientes, cada tarea tiene mapeado el kernel y sus páginas para código, datos pila LVL 3 y datos pila LVL 0.

IDEA puedo tener los códigos en el área kernel, el cual estaría mapeado por "identity mapping" US=0. Accedería a través de una interrupción syscall con nivel de privilegio 0. Guardo entonces en una variable global ~~de tc~~ un vector codigos[3] con las direcciones de las pág de códigos.

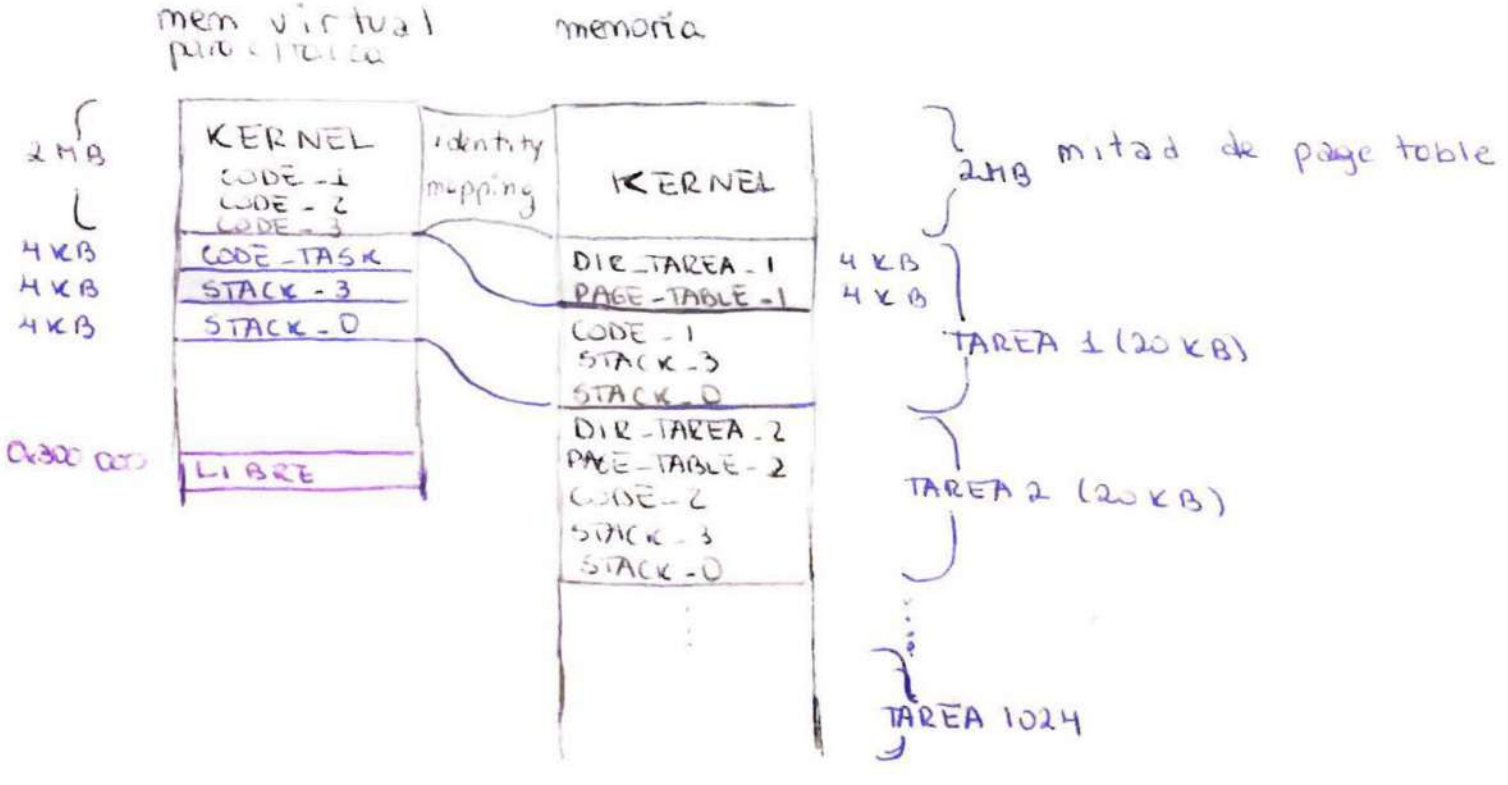
LVL 3 task ——→

LVL 0 new-task ——→ escribe en EAX (sin alterar el resto de registros, al final del popad hago mov EAX, □)

¡leer EAX!

→ setear TSS de la nueva tarea

Mapa de memoria



Rango de direcciones

Kernel 0x00000000 a 0x001FFFFF (2 MB)

código tarea 0x00200000 a 0x2007FF (4 KB)

dato tarea pila 3 0x00201000 a 0x201FFF (4 KB)

dato tarea pila 0 0x00202000 a 0x2027FF (4 KB)

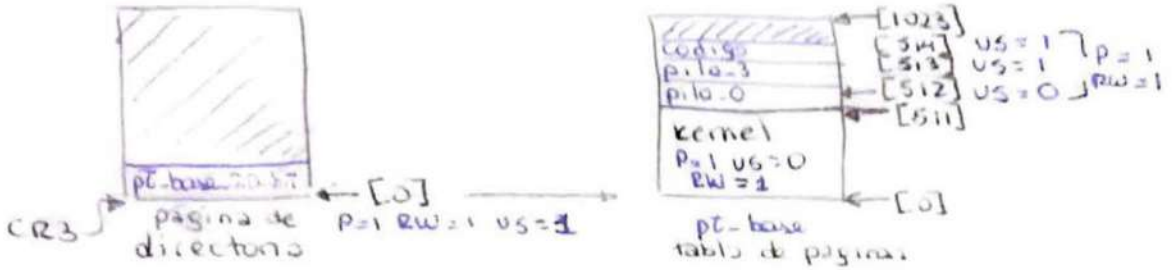
Para una tarea i, $i \in \{1 \dots 1024\}$, las páginas que se mapean son:

$2\text{MB} + (i-1) \cdot 20\text{KB} + 8\text{KB}$ hasta $2\text{MB} + (i+1) 20\text{KB} - 1$

$0x00200000 + (i-1) \cdot 0x5000 + 0x2000$ hasta

$0x00200000 + (i+1)\, 0x5000 - 0x0001$.

Esquema de paginación (escrito en en decimal)



$0x00200000 + (i-1)\, 0x5000$ directorio CR3

$0x00200000 + (i-1)\, 0x5000 + 0x1000$ pt-base

$0x00200000 + (i-1)\, 0x5000 + 0x2000$ código

$0x00200000 + (i-1)\, 0x5000 + 0x3000$ pila LVL 3

$0x00200000 + (i-1)\, 0x5000 + 0x4000$ pila LVL 0.

P = 0 (no mapeado)

Dirección libre de la tarea 0x300000 para hacer copia del código para la nueva tarea del tipo pasado por parámetro.

(b) Para la syscall "new_task" DPL = 3 (índice 41 porque pinto)

Para excepciones (índice [0] a [31] en IDT): DPL = 0

" interrupción del reloj (índice [32]) : DPL = 0

Todas con sel_CODE_RUTINA: SEL_CODE_LVL_0, tipo 0xE y P = 1. El resto de las entradas con P = 0.

Ejercicio 2

a) EN ASM

```
extern nuevaTarea

_isr41:
    pushad
    mov edi, cr3
    push edi
    push eax
    call nuevaTarea
    add esp, 8
    cmp eax, 0
    je .huboNuevaTarea
    popad
    xor eax, eax
    dec eax          ; -1 si no hubo nueva tarea
    iret

.huboNuevaTarea:
    popad            ; devuelvo el tipo de tarea creada
    iret             ; que yo me vine en eax
```

EN C

```
#define INT_ON 0x202
#define VIRTUAL_FREE 0x300000
#define U 1
#define S 0
uint32_t nuevaTarea(uint32_t eax, uint32_t cr3){
    tss* new_TSS = tss_getfree();
    if (new_TSS == NULL){
        return -1;
    }

    uint8_t new_ID = Buscar_ID(new_TSS);
    mmu_mapPage((pde*) cr3, VIRTUAL_FREE, 0x202000 + new_ID * 0x5000, 1, 1)
    copiar_mem(codigos[eax-1], VIRTUAL_FREE);
    pde* new_cr3 = mmu_newPD();
    mmu_mapKernel(new_cr3);
```

```
    mmu_mapPage(new_cr3, CODIGO, 0x202000 + new_ID * 0x5000, 4, 1);
    mmu_mapPage(new_cr3, PILA_3, 0x203000 + new_ID * 0x5000, 4, 1);
    mmu_mapPage(new_cr3, PILA_0, 0x204000 + new_ID * 0x5000, 5, 1);
    desmapear((pde*) cr3, VIRTUAL_FREE);

    new_TSS->cr3 = new_cr3;
    new_TSS->edx = sched_getID();
    new_TSS->eip = CODIGO
    new_TSS->esp = PILA_3,
    new_TSS->esp0 = PILA_0,
    new_TSS->EFLAGS = INT_ON,
    new_TSS->CS = SEL_CODE_LVL_3,
    new_TSS->SS0 = SEL_DATA_LVL_0;
    new_TSS->DS = SEL_DATA_LVL_3;
      "      GS = "      "      ;
      "      ES = "      "      ,
      "      FS = "      "      ;
    new_TSS->IOMAP = 0xFFFF;
    // el resto de atributos pueden comenzar con 0 o cualquier cosa
    sched_add(new_TSS);
    return 0;
}

void mmu_mapKernel(pde* new_cr3){
    for(uint16_t i=0; i<512; ++i){
        mmu_mapPage(new_cr3, i, i, 5, 1);
    }
}

void copiarMem(uint32_t* src, uint32_t* dest){
    for(uint16_t i=0; i<1024; ++i){
        dest[i] = src[i];
    }
}
```

sched.c Buscar ID (tss* new_TSS) recorre el array de tareas y retorna el índice de la posición cuya TSS coincida con el parámetro.

Ejercicio 3

(a) Supongamos que tenemos N tareas, con ID desde 0 a N-1. La rutina de atención de interrupción de reloj debe

1) llamar al PIC para avisar que atendió la int
2) llamar a la función que indica la sig tarea a ejecutarse
3) Verificar si no es la tarea actual, en cuyo caso salta, de lo contrario sigue la actual

EN ASM

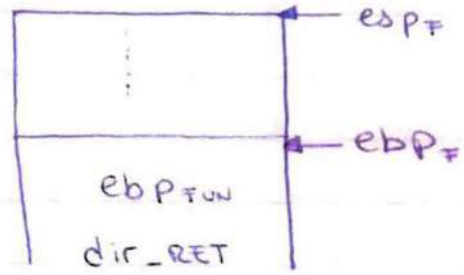
```
extern logEIP, para item (b)
global _isr32
extern fin_intr_pic
extern next_task_TSS

offset  DD 0
selector DW 0


_isr32
    pushad
    call fin_intr_pic
    call next_task_TSS
    ltr cx
    cmp cx, ax
    je .sigoYo
    mov [selector], ax
    jmp far [offset]
.sigoYo
    popad
    iret
```

(b) En primer lugar, el índice en la GDT de la tarea en ejecución lo puedo obtener del selector de TSS que se encuentra cargado en el Task Register. Tengo

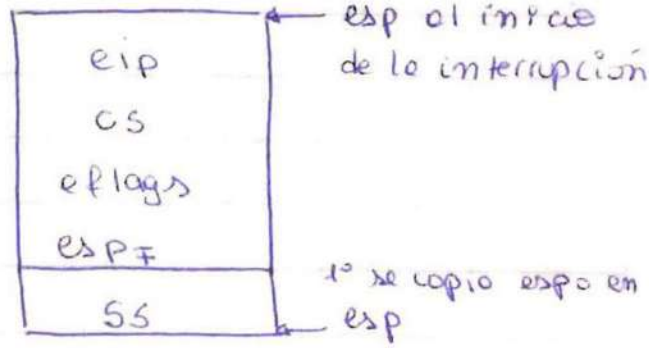
que shiftear 3 lugares a la derecha pues hay 2 bits del RPL y 1 del TI.
Luego la función la podemos obtener con la dirección de retorno pusheada en la pila de nivel 3, habría que restarle 4 bytes para obtener el puntero. (Esta dirección la pusheó el call). Como la función respeta convención C, sabemos que hizo:

```
Fun  push ebp
     mov ebp, esp
     :
     INTERRUPCION
     :
     pop ebp
     ret
```

De modo que la pila 3



Si tuviéramos segmentación flat podríamos leer directamente con el ebp que llega a la interrupción del reloj, pues con privilegio 0 accedo al seg de datos LVL 3. Como el enunciado no aclara esto, por las dudas vemos o buscar el selector de datos de nivel 3 pusheado en la pila de nivel 0 al producirse la interrupción.



EN ASM; cambio de privilegio

```
/ define offset_pila_ss 96
_isr32:
    pushad          , preserve reg
    call fin_intr_pic , avisa al PIC
    add rbp, 8      , busco dir_RET
    mov si, [esp + offset_pila_ss]
    mov es, si
    mov edi, [es : ebp] ; leo con sel_datos_3
    xor ecx, ecx
    ltr cx
    shr ecx, 3
```

```
    push edi
    push ecx
    call logEIP
    add esp, 8
    call next_task_TSS
    ltr cx
    cmp cx, ax
    je .sigoYo
    mov [selector], ax
    jmpfar [offset]
.sigoYo
    popad
    iret
```