Ejercicio:

- En un sistema similar al que implementamos en los talleres del curso (modo protegido con paginacion activada), se quiere implementar un servicio tal que cualquier tarea del sistema lo pueda invocar mediante la siguiente instruccion:

  int 0x100

Recibira los siguientes parametros (en ese orden):

- uint32_t virt, una direccion de pagina virtual
- uint32_t phy, una direccion de pagina fisica
- uint16_t task_sel, un selector de segmento que apunta a un descriptor de TSS en la GDT

El servicio en cuestion forzara la ejecucion de codigo comenzando en la direccion fisica phy, mapeado en virt. Tanto la tarea actual como la tarea que sera pasada como parametro (indicada por su task_sel) deben realizar la ejecucion de la "pagina" fisica en cuestion. Para eso, dicho servicio debera:

- i. Realizar los mapeos necesarios
- ii. Modificar los campos necesarios para que la tarea determinada por task_sel, retome su ejecucion en la posicion establecida la proxima vez que se conmute a ella.
- iii. Modificar los campos necesarios para que la tarea actual, retome su ejecucion en la posicion establecida una vez completada la llamada

Se pide:

a. Dar un ejemplo de invocacion de dicho servicio b. Definir o modificar las estructuras de sistema necesarias para que dicho servicio pueda ser invocado c. Implementar dicho servicio (pseudocodigo)

Detalles de implementacion:

- El codigo en cuestion a donde se salta es de nivel 3.
- Los punteros a las pilas de nivel 3 de ambas tareas y el puntero a la pila de nivel 0 de la tarea (nivel 3) pasada por parametro, deberan ser reinicializados a la base de la pila, teniendo en cuenta que las mismas comienzan al final de la pagina y no se extienden mas que 4096 bytes.
- Asumir que todas las tareas ya fueron alojadas al menos una vez y que el cambio de tareas se hace en la rutina de interrupcion de reloj, como en el taller

---

Resolucion:

Ejemplo de invocacion de la syscall 0x100 en algun punto de la tarea que esta corriendo:

```
    .
    .
    .
mov eax, virt
mov ebx, phy
mov cx, task_sel
int 0x100
    .
    .
    .
```

En el archivo "isr.asm" se deberia definir el siguiente codigo:

```
global _isr100

_isr100:

    ;uint32_tid forzar_ejecucion(uint32_t virt, uint32_t phy, uint16_t task_sel)

    push ecx  ; cx  <- uint16_t task_sel, un selector de segmento que apunta a un descriptor de TSS en la GDT
    push ebx  ; ebx <- uint32_t phy, una direccion de pagina fisica
    push eax  ; eax <- uint32_t virt, una direccion de pagina virtual

    call forzar_ejecucion

    ; iii - Para que esta corriendo, retome su ejecucion en la posicion establecida una vez completada la llamada:

    ; return eax <- virt
    ; Estado de la pila nivel 0: [EAX|EBX|ECX|EIP|CS|EFLAGS|ESP|SS]
    mov [esp + 12], eax     ; EIP <- Virt fuerza la ejecucion a la direccion virtual
    ; Reinicio la pila de nivel 3 de la tarea que esta corriendo
```

```
    mov ecx, [esp + 24]     ; busco el esp de nivel 3 en la pila de nivel 0.
    and ecx, 0xFFFFF000
    add ecx, 0x1000         ; reinicia la pila de nivel 3, poniendo a cero los primeros 3 nibbles y sumando 0x1000
    mov [esp + 24], ecx     ; no hace falta cambiar el ss, porque no hubo cambio de contexto.

    ; arregla la pila para que se pueda hacer el irret, podrian haber tocado directamente el esp a mano: add esp, 12
    pop eax
    pop ebx
    pop ecx

    ; Estado de la pila nivel 0 aqui: (direcciones menores) [EIP|CS|EFLAGS|ESP|SS] (direcciones mayores)
    iret  ; y esto reestablece la pila de nivel 0
```

--

```
#define MMU_P (1 << 0)     /* pagina presente */
#define MMU_W (1 << 1)     /* escritura */
#define MMU_R (0x0)        /* lectura */
#define MMU_U (1 << 2)     /* nivel usuario */

uint32_t forzar_ejecucion(uint32_t virt, uint32_t phy, uint16_t task_sel)
{
    //i - MAPEOS:

    // Obtener CR3 de ambas tareas, la que esta en ejecucion y la que pasa por parametro:

    uint32_t cr3_running_task = rcr3();         /* CR3 de la tarea que esta corriendo, esta definida en "i386.h" */

    // Obtengo el indice en la GDT de la TSS para el selector de segmento de la tarea que me pasan por parametro
    uint_16 index_task_param = task_sel > 3  /* task_sel = [INDEX (13 bits) | TI (1 bit) | RPL (2 bit)] */

    uint_32 tss_address = (gdt[index_task_param].base_31_24 < 24) |
                          (gdt[index_task_param].base_23_16 < 16) |
                          (gdt[index_task_param].base_15_0);

    tss* tss_param_task = (tss*)tss_address;
        uint_32 cr3_param_task = tss->cr3;      /* CR3 de la tarea que me pasan por parametro */

    // De su taller pueden usar: mmu_mapear_pagina(uint32_t cr3, uint32_t virtual, uint32_t fisica, uint32_t attrs)
    mmu_mapear_pagina(cr3_running_task, virt, phy, MMU_U | MMU_R | MMU_P);      /* nivel usuario, lectura y que este presente */
    mmu_mapear_pagina(cr3_param_task,   virt, phy, MMU_U | MMU_R | MMU_P);      /* nivel usuario, lectura y que este presente */


    // ii - Para que la tarea pasada por parametro retorne en la posicion establecida la proxima vez que se conmute a ella:

    /* fuerza la ejecucion a partir de virt de la tarea que me pasan por parametro */
    tss_param_task->eip = virt;

    /* La tarea que pasan por parametro esta frenada en la interrupcion de reloj, es decir,
       que el CS es de nivel 0. En la interrupcion de reloj el unico lugar donde las tareas
       quedan desalojadas y para poder forzar la ejecucion del codigo que me pasan por
       parametro tengo que hacer cs:eip, sino hago esto estaria haciendo cs0:eip y esta mal. */

    tss_param_task->cs = GDT_CS_RING_3;         /* cargo selector de segmento de codigo nivel 3 */

    /* Reinicializo el puntero de pila nivel 0 y 3 de la tarea que me pasan por parametro,
       poniendo en cero los primeros doce bits y sumando 0x1000. */

    /*
        IMPORTANTE: Desde nivel 0 NO podemos usar la pila de nivel 3 para
        guardar el estado de retorno y variables locales. Por lo tanto, se
        debe intercambiar la base de la pila. La nueva base de la pila
        se toma desde los campos SS0:ESP0 en la TSS. El estado de la pila
        de nivel 3 se guarda en la pila de nivel 0.
    */

    /*
        Estado de la pila dentro del handler de interrupcion de reloj para la tarea
        que me pasan por parametro:
        (direcciones menores) [EAX|ECX|EDX|EBX|ESP|EBP|ESI|EDI|EIP|CS|EFLAGS|ESP|SS] (direcciones mayores)
    */

    uint32_t* esp_task_param =  (uint32_t*)(tss_param_task->esp);           /* obtiene esp de nivel 0 */
    tss_param_task->esp      = esp_task_param[11] & (0xFFFFF000);           /* reinicia la pila de nivel 3. Mirar EJEMPLO al final
*/
    tss_param_task->esp     += 0x1000;
    tss_param_task->ss       = GDT_IDX_DS_RING_3;                          /* selector de segmento de pila tiene que ser de datos
nivel 3*/
```

```
    /* Las dos siguiente lineas podria no haberse hecho, porque son campos estaticos en la TSS
       Esto quiere decir, que una vez que fueron seteados, el procesador ya no puede modificarlos y
       y el puntero de pila que quedo guardado en la TSS ya se encuentra bien inicializado */

    tss_param_task->esp0     = (uint32_t)(esp_task_param) & (0xFFFFF000)    /* reinicia la pila de nivel 0 */
    tss_param_task->esp0    += 0x1000;

    return virt;
}
```

EJEMPLO para convencerse que lo que estamos haciendo verdaderamente reinicia la pila:

- Por ejemplo que se haya pedido: Un pila en la direccion cuyo marco de pagian esta en:

```
            0x00134000
```

luego la ultima direccion valida para la pila es:

0x00134FFF

Por lo tanto, al inicio, el ebp y esp (base y tope de la pila) se encuentran ambos inicializados en el valor:

```
            ebp, esp = (0x00135000)
```

Supongamos ahora que la pila se estuvo usando (para poder reinicializarla), por lo que se modifico esp a:

```
            ebp =  0x00135000
            esp =  0x00134fe0 (esp < ebp porque la pila aumenta hacia las direccion menores en memoria)
```

Para reiniciarla hacemos los primeros doce 12 bits (o primeros 3 nibbles) a cero y sumamos 0x1000: 0x00134fe0 ---> 0x00134000 ---> 0x00135000