Inciso 1

Sería necesario que en la IDT hayan entradas para atender a estos dos servicios. Cualquier índice que no entre en conflicto con los definidos en la arquitectura pueden funcionar. Tomaremos los índices 100 y 123. En esas entradas de la IDT, el campo offset apuntará a los handler definidos como _isr100 y _isr123. El campo segsel tendrá el valor de un selector de código de nivel 0 con RPL 0 y el campo de atributos corresponde a un descriptor de interrupciones de 32 bits con DPL de usuario (0xEE00) para que las tareas puedan llamar a los servicios.
También definiremos en la GDT una descriptor de TSS en el índice 18 similar a los decriptores anteriores, que apunte a un struct de tipo tss_t llamado tss_registrada. Esta tss también es muy similar a las de las tareas, pero defiere en los siguientes puntos:

- esp = 0x540FA00. Este valor es el comienzo de la pila de la tarea, justo cuando termina la última página de del área de datos

Cuando una tarea registra una funcion seteamos una flag global en 1, guardamos su puntero en el campo eip de tss_registrada, pedimos una página con kMallocPage y la mapeamos en la direccion virtual 0x540FA00 para que la función pueda utilizar esa página para su pila de nivel 3. Finalmente seteamos el cr3 de tss_registrada igual al cr3 de la tarea actual.
Luego en la próxima interrupción de reloj, la función sched_next_task notará que hay una función registrada, por lo que devolverá el selector de tss en el índice 18 de la GDT. Es importante que la función registrada se encargue de llamar a la syscall regresar. Esta syscall se encarga de “desregistrar” la función, liberar y desmapear la memoria que habíamos reservado para la pila. Por último saltamos a la próxima tarea a ejecutarse.

```
extern registrar
extern regresar
global _isr100
_isr100:
      pushad
      push eax
      call registrar
      add esp, 4
      popad
      iret
global _isr123
_isr123:
      pushad
      call regresar
      call sched_next_task
      mov word [sched_task_selector], ax
      jmp far [sched_task_offset]
      popad
      iret
```

```
#define VIRT_STACK_START 0x540FA00

int8_t curr_idx = 0;
uint32_t funcRegistrada = 0;
uint32_t physical;

uint16_t sched_next_task(void) {
  if(funcRegistrada){
      return (18 << 3) | GDT_RPL_RING_0
  }
  curr_idx = (curr_idx + 1) % 4;
  return tasks[curr_idx].idx_gdt;
}

void regresar() {
  funcRegistrada = 0;
  kFreePage(physical);
  mmu_unmap_page(TSSs[curr_idx].cr3, VIRT_STACK_START - PAGE_SIZE)
}

void registrar(uint32_t func) {
  funcRegistrada = 1;
  physical = kMallocPage();
  mmu_map_page(TSSs[curr_idx].cr3, VIRT_STACK_START - PAGE_SIZE,
physical, 1, 1);
  tss_registrada.cr3 = TSSs[curr_idx].cr3;
  tss_registrada.eip = func;
  tss_registrada.esp = VIRT_STACK_START;
}
```

Inciso 2:

```
uint32_t global = 0;

void TareaA()
{
   __asm__ volatile("int $100"
                    : "a"(&aux),
                    : "memory",
                    "cc");
  while (1){}
}

void aux()
{
  ++global;
  __asm__ volatile("int $123"
                    : "memory",
                    "cc");
}
```