| Nº Orden | Apellido y nombre | L.U. | Cantidad de hojas |
|---|---|---|---|
| | | | |

# Organización del Computador 2
# Segundo parcial – 29/06/17

| 1 (30) | 2 (40) | 3 (30) | |
|---|---|---|---|
| | | | |

*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. (30 puntos)

Se tiene la siguiente tabla GDT:

| Indice | Base | Límite | DB | S | P | L | G | DPL | Tipo |
|---|---|---|---|---|---|---|---|---|---|
| 0x32 | 0x8F8A9000 | 0x001FF | 1 | 1 | 1 | 1 | 1 | 3 | 0xA |
| 0x41 | 0x00323000 | 0x00FFF | 1 | 1 | 1 | 1 | 1 | 3 | 0x2 |
| 0x73 | 0x03123000 | 0x0FFFF | 1 | 1 | 1 | 1 | 1 | 0 | 0x8 |
| 0x92 | 0x10AB8000 | 0x00000 | 1 | 1 | 1 | 1 | 1 | 0 | 0x2 |

Y el siguiente esquema de paginación:

| Rango Lineal | Rango Físico | Atributos |
|---|---|---|
| `0x00303000` a `0x00C0CFFF` | `0x3F532000` a `0x3FE3BFFF` | read/write, level 0 |
| `0x03210000` a `0x10AFFFFF` | `0x23C16000` a `0x31505FFF` | read/write, level 3 |
| `0x8F800000` a `0x8FFFFFFF` | `0xB56A7000` a `0xB5EA6FFF` | read/write, level 3 |

(12p) a. Especificar todas las entradas de las estructuras necesarias para construir un esquema de paginación. Suponer que todas las entradas no mencionadas son nulas.

(18p) b. Resolver las siguientes direcciones, de lógica a lineal y a física. Utilizar las estructuras definidas y suponer que cualquier otra estructura no lo está. Si se produjera un error de protección, indicar cuál error y en qué unidad. Definir `EPL` en todos los casos. El tamaño de todas las operaciones es de 4 bytes.

I - `0x0190:0x00003221` - CPL `11` - `lectura`

II - `0x0208:0x000913AF` - CPL `11` - `lectura`

III - `0x0398:0x00123831` - CPL `00` - `ejecución`

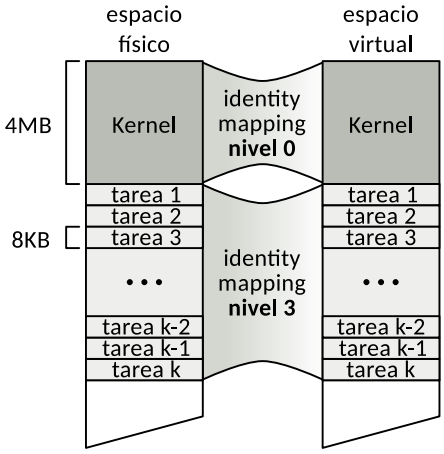
IV - `0x0490:0x00000012` - CPL `00` - `escritura`

V - `0x0190:0x00001021` - CPL `00` - `ejecución`

VI - `0x0208:0x00833414` - CPL `10` - `lectura`

## Ej. 2. (40 puntos)

Se tiene un sistema en modo protegido, con paginación activa. El mismo ejecuta `K` tareas una a una en orden, con un máximo de 500 tareas. Todas las tareas se encuentran alojadas a un área mapeada con *identity mapping* en la cual cualquier otra tarea puede acceder como muestra la siguiente figura:




Cada tarea tiene asignado exactamente 8 KB de espacio direccionable para código, datos y pila. El sistema ejecutará las tareas y detectará cuando una tarea modifica el área de memoria perteneciente a otra tarea, es decir, accede fuera de sus 8 KB. En el caso de detectar una modificación, el sistema ejecutará la función `void fue_modificada(uint32_t tareaModificada, uint32_t tareaModificadora)` indicando el número de la tarea que fue modificada y el número de la tarea que la modifico. Considerar que esta función debe ser ejecutada en algún momento antes de volver a ejecutar la tarea que fue modificada.

(10p) 1. Indicar los campos relevantes de todas las estructuras involucradas en el sistema para administrar segmentación, paginación, tareas, interrupciones y privilegios. Instanciar las estructuras con datos y explicar su funcionamiento. Describir tanto el esquema de segmentación como el de paginación.

(20p) 2. Programar en ASM/C la rutina de atención de interrupciones del reloj. Recordar que debe intercambiar las tareas y detectar modificaciones en memoria.

(10p) 3. ¿Es posible detectar que posición de memoria de una tarea fue modificada por otra? De ser posible, describir en detalle como implementaría este mecanismo o justificar por que no es posible.

Nota: Considerar el uso de los bits **Dirty** y **Accessed** en las *Page Table Entry*

# Ej. 3. (30 puntos)

En un sistema tipo con segmentación y paginación activa, se ejecutan concurrentemente tareas denominadas gusanos. Estos pueden multiplicarse mendiante un servicio del sistema. Las tareas llaman al servicio `multiplicate` y este, crea dos copias de la tarea, una de nombre *derecha* y otra *izquierda*. Para identificar cual es izquierda y cual derecha, el servicio guarda en el registro `al` el byte "I" o "D" respectivamente.

Para duplicar tareas, el servicio cuenta con la función: `tss* duplicar(tss*)`, que toma un puntero a la `tss` de una tarea y retorna una copia de la tarea, pero en otro espacio de memoria físico.

(5p) 1. Explicar detalladamente que información debe copiar la función `duplicar`.

(20p) 2. Programar en ASM/C la rutina de atención del servicio `multiplicate`.

(5p) 3. ¿Es posible que las tareas duplicadas compartan el mismo `CR3`? ¿Qué problemas generaría?

Nota: Considerar que la información de la `tss` de la rutina que llama al servicio del sistema contiene la información de la ultima vez que esta fue desalojada.