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

Organización del Computador 2
Segundo parcial – 27/06/2013

1 (40)	2 (40)	3 (20)	

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.
- Existen tres notas posibles para los parciales: 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. Para los recuperatorios existen sólo dos notas posibles: I: 0 a 64 pts y A: 65 a 100 pts.

Ej. 1. (40 puntos)

1. (10 puntos) Describir cómo se completarían las entradas de la GDT en función de los segmentos que se detallan en la siguiente tabla. Los valores base y límite deben indicarse en hexadecimal.

Indice	Desde	Tamaño	Permisos	Tipo
3	1Mb	3Mb	`level 1`	Código - lectura
6	10Kb	1b	`level 0`	Datos - lectura/escritura
9	0	3.5Gb	`level 3`	Código - solo ejecución
14	2Gb	2Gb	`level 2`	Datos - lectura

2. (13 puntos) Especificar todas las entradas de las estructuras necesarias para construir un esquema de paginación según la siguiente tabla. Suponer que todas las entradas no mencionadas son nulas. Los rangos incluyen el último valor. Los permisos deben definirse como usuario.

Rango Lineal	Rango físico
`0x00002800` a `0x00004800`	`0x00019000` a `0x0001B800`
`0x00000000` a `0x00001800`	`0xC0000000` a `0xC0001800`

3. (13 puntos) Resolver las siguientes direcciones, de lógica a lineal y a física. Utilizar las estructuras definidas en los ítems anteriores y suponer que cualquier otra estructura no está definida. Si se produjera un error de protección, indicar cuál error y en qué unidad (definir EPL en todos los casos).

   - `0x004B:0x00000001` - CPL `11` - `lectura`
   - `0x0030:0x00100000` - CPL `00` - `lectura`
   - `0x001A:0x00000400` - CPL `01` - `lectura`
   - `0x0002:0x00001000` - CPL `10` - `lectura`
   - `0x0018:0x05000060` - CPL `00` - `ejecución`
   - `0x004B:0x00001000` - CPL `10` - `ejecución`

4. (4 puntos) Cuál es el efecto que se produce en el sistema de traducción de memoria (de direcciones virtuales a físicas) en un SO si no se habilita la línea A20? A que dirección física se traduce la última entrada del punto 1.3?

Ej. 2. (40 puntos)

Se cuenta con un kernel básico al cual se le quiere extender las funcionalidades de su `MMU`. Actualmente, las tareas (que corren en nivel 3) tienen un área de memoria (virtual) reservada para datos (que pueden usar para leer y escribir). La misma está comprendida, inicialmente, entre `DATA_START` y `DATA_END`. Se desea implementar una llamada al sistema que permita incrementar el tamaño del área de acuerdo a un valor pasado como parámetro.

La *syscall* toma el parámetro `incremento` en el registro `eax`. La descripción exacta de la funcionalidad es la siguiente: Incrementa el área de datos la cantidad de bytes indicada por `incremento` (es múltiplo de 4Kb) y retorna la dirección del nuevo límite. Siempre trata de asignar la cantidad pedida, en caso de no poder hacerlo, asigna la mayor cantidad posible. Si no puede asignar nada, retorna el límite actual. Un ejemplo de uso sería el que se muestra abajo. En este caso, la llamada retorna en `eax` el valor `DATA_END + 0x5000`.

```
mov eax, 0x5000
int 95
```

Se pide:

a) (10 puntos) Agregar un descriptor de *Interrupt Gate* en la `IDT` para la *syscall* en cuestión.
b) (5 puntos) Describa las estructuras de datos necesarias para poder implementar la llamada y decir con qué valores deben estar inicializadas antes de poder usarla.
c) (25 puntos) Implementar el *handler* de la llamada al sistema (`_isr95`).

Para llevar a cabo la implementación, cuenta con las siguiente funciones:

- `unsigned int mmu_dame_fisica_libre_kernel()`: Retorna la dirección de una página física libre para datos de kernel. **Puede** asumir que siempre que llame a esta función retornará una página disponible.
- `unsigned int mmu_dame_fisica_libre_usuario()`: Retorna la dirección de una página física libre para datos de usuario. En caso de que no haya una página disponible, la función retorna `0xFFFFFFFF`.
- `unsigned int leer_cr3()`: Retorna el contenido del registro `cr3`.
- `unsigned int tarea_pid()`: Retorna el identificador de la tarea actual (número entre 0 y `CANT_MAX_TAREA`).

Notas:

- Recomendación: Divida la implementación en funciones convenientes. Puede escribirlas en C y/o ASM según más le parezca.
- La cantidad máxima de tareas que puede manejar el kernel es `CANT_MAX_TAREAS`

Ej. 3. (20 puntos)

En una arquitectura sin reloj, se desea implementar un sistema operativo ejecute dos tareas S (system) y U (user) concurrentemente. La tarea S debe correr con nivel de privilegio 0 y la tarea U con nivel 3.

a) (7 puntos) Describir un esquema de software que implemente un sistema así, mencionando la idea general, la forma en que se cambia el privilegio y las rutinas necesarias. ¿cómo se realizaría el task switch?¿Es necesario en su implementación que S y U realicen alguna acción para realizarlo?

b) (7 puntos) Dado el punto anterior, indique que estructuras debería poseer el sitema. Además, indique que entradas habría que agregar a las tablas del sistema, indicando el valor de sus campos.

c) (6 puntos) Escriba en ASM las secciones de código relativo al task switch de los componentes antes mencionados.