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

Organización del Computador 2
Segundo parcial – 24 de Junio de 2014

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.
- 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)

a) (10 puntos) Describir cómo se completarían las entradas de la GDT en función de los segmentos que se detallan en la siguiente tabla. Indicar los campos `Indice`, `Base`, `Limite`, `S`, `G`, `D/B`, `AVL`, `L`, `P`, `DPL` y `Tipo`. Los valores de `Base` y `Limite` deben indicarse en hexadecimal.

Indice	Desde	Tamaño	Permisos	Tipo
3	16 Kb	65540 bytes	`nivel 0`	Código - lectura
4	4 Kb	64 Mb	`nivel 3`	Código - solo ejecución
6	1 Gb	1 Gb	`nivel 0`	Datos - lectura
7	2 Gb	2 Gb	`nivel 3`	Datos - lectura/escritura

b) (10 puntos) Especificar todas las entradas de las estructuras necesarias para construir un esquema de paginación particular según la siguiente tabla. Suponer que todas las entradas no mencionadas son nulas. Los rangos incluyen el último valor.

Rango Lineal	Rango Físico	permisos
`0x0000D000` a `0x00011FFF`	`0x1000E000` a `0x10012FFF`	usuario
`0xCA050000` a `0xCA050FFF`	`0x1000E000` a `0x1000EFFF`	root

c) (10 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.

I - `0x18:0x00008FFE` - CPL `00` - `ejecución`

II - `0x33:0x0000A123` - CPL `00` - `lectura`

III - `0x38:0x4A050ABA` - CPL `11` - `lectura`

IV - `0x23:0x0000C000` - CPL `11` - `ejecución`

V - `0x3B:0x8A050800` - CPL `11` - `escritura`

recordar:
`64B=0x40, 4KB=0x1000, 65536B=64KB, 1MB=0x100000,`
`1GB=0x40000000, 2GB=0x80000000, 3GB=0xC0000000, 4GB-1=0xFFFFFFFF.`

Ej. 2. (35 puntos)

Los muchachos de Sabella están entrenando para el partido de mañana contra Nigeria, y aquí preparamos el simulador donde se llevará a cabo el precalentamiento. El mismo es sencillo, los jugadores se mueven dentro de la cancha en distintas direcciones.

Cada jugador es una tarea corriendo en nivel 3. El sistema brinda la syscall 86: MOVERSE, que recibe en AL el código de la dirección (arriba/abajo/izquierda/derecha/mantienepos). Se otorga 1 clock de reloj a cada jugador para que elija su dirección (puede hacerlo **tantas veces como quiera** y sólo se tendrá en cuenta la última elección). Si el jugador no hace ninguna llamada se considera que permanece en el mismo lugar. Después de que se cicló por todos los jugadores se ejecuta a una tarea *todopoderosa* en nivel 0, que mueve a los jugadores en consecuencia (también tiene 1 ciclo para hacerlo). Esta tarea **debe resetearse antes de volver a ser ejecutada**. Después de la todopoderosa se vuelve a comenzar el ciclo. Se pide:

a) (5 puntos) Detallar las entradas de la `IDT` necesarias para implementar el simulador.

b) (5 puntos) Describir las estructuras de datos necesarias para poder implementar MOVERSE y la ISR del reloj. Definir con qué valores deben estar inicializadas antes de poder usarla.

c) (10 puntos) Implementar el *handler* de la llamada al sistema (`_isr86`).

d) (15 puntos) Implementar el *handler* de la interrupción de reloj

Los handlers deben implementarse en ASM. Es posible usar C para funciones auxiliares, pero no en pseudocódigo.

Nota: El código de los jugadores y el todopoderoso se asume dado, pero **usted debe definir** cómo se pasan entre sí los movimientos en los ítems b), c) y d). También se cuenta con la función `tss_copy(tss *destino, tss *fuente)` que copia una tss entera `fuente` en la tss `destino`.

Ej. 3. (35 puntos)

En Orga2 estamos construyendo un sistema operativo con debugger incorporado, que se activa cuando una tarea causa una excepción. Para tal fin, los handlers de las interrupciones del 0 al 31 hacen lo siguiente:

```
_isrHandler:
jmp far SELECTOR_DEBUGGER:0
```

donde SELECTOR_DEBUGGER es el selector de la GDT de una tarea debugger. Se desea implementar en C las siguientes funcionalidades para el debugger:

a) (20 puntos) una función `unsigned int valor_memoria(void *direccion, tss *tarea_a_debuggear)`, que devuelve el valor de memoria de 32-bits en `dirección` en el espacio de la `tarea_a_debuggear`. Puede asumir lo que le parezca más conveniente respecto a segmentación.

b) (10 puntos) usando la función anterior, la función `unsigned int tope_pila(tss *tarea_a_debuggear)`, que devuelve el contenido en el tope de la pila de la `tarea_a_debuggear` en el instante antes de la interrupción.

c) (5 puntos) ¿Asumió algo sobre la segmentación en los puntos a) y b)? En caso de que la respuesta sea afirmativa indicar qué y para qué.

El debugger tiene mapeado con identity mapping el kernel pero no toda la memoria. Tenga en cuenta que estas dos funciones correrán en el contexto de la tarea debugger

Se cuenta con las siguientes funciones dadas:

- `cr3()`, que devuelve el valor actual del registro `cr3`.
- `dame_virtual_libre()` que devuelve una dirección virtual libre y alineada a 4kb dentro del espacio del debugger.
- `dame_fisica_libre()` que devuelve la dirección física de una página libre alineada a 4kb.