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

Organización del Computador 2
Recuperatorio del segundo parcial – 12/07/2011

1 (45)	2 (45)	3 (10)	

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. (45 puntos)

1. (20 puntos) Construya un esquema de segmentación en dos niveles (0 y 3), que evite que los programas de usuario modifiquen el código del sistema operativo, su propio código, o el código de otros programas. El sistema operativo, sin embargo, deberá poder cargar estos programas en memoria; para esto necesita poder escribir en el segmento de código de usuario. Considere que no se requieren más de 128 MB de espacio para el código de todos los programas de usuario, que estos programas no utilizarán más de 64 MB de espacio para sus datos, y que el código del kernel requiere 32 MB para ejecutarse.

   Escriba los descriptores de segmento necesarios para implementar su esquema de segmentación. Liste los campos *base* y *límite* en hexadecimal, y aclare los flags que configura en cada segmento. Considere que el sistema posee 1 GB de RAM, y que no es necesario proteger los datos de los distintos programas de usuario entre sí, ni preocuparse por la pila.
2. (25 puntos) El programa de usuario "Segundo recuperatorio de Orga2.exe" está cargado en memoria física a partir de la dirección `0x40000000`. Sin embargo, está compilado y linkeado estáticamente para ejecutarse a partir de la dirección virtual `0x8000000`. El código del programa ocupa 4 páginas de 4 KB consecutivas. La pila del programa está configurada inicialmente en la dirección virtual `0xA0000000`, y no ocupará nunca más de 8 KB. Se tiene disponible el rango de memoria física `[0xD0000000, E0000000]` para la pila, y el rango de memoria física `[0x10000000, 0x20000000]` para el directorio y las tablas de páginas.

   Construya un esquema de paginación que permita ejecutar este programa tal y como fue compilado. Suponga que se está utilizando segmentación flat. Describa el directorio y las tablas de páginas necesarias. Liste todas las entradas de directorio y/o tablas que utilice, indicando todos los campos de cada entrada, incluyendo los flags. Puede suponer que cualquier rango de direcciones no mencionado es nulo.

Ej. 2. 45 puntos

La cátedra ha decidido actualizar el sistema operativo Orga2SO para que disponga de un scheduler con niveles de prioridad. Este nuevo scheduler manejará dos niveles de prioridad: **alta** y **baja**. Las tareas a ejecutar se mantendrán en un arreglo de longitud fija, y su nivel de prioridad será indicado por uno de los campos de la estructura contenida en este arreglo. Mientras haya tareas en el arreglo con prioridad alta, no se ejecutarán tareas con prioridad baja. Las tareas se ejecutan en orden, interpretando el arreglo como un arreglo circular.

Cada tarea se ejecuta, como máximo, una cantidad de ticks identificada por la variable `QUANTUM`. Si una tarea con prioridad alta llega a ejecutarse durante `QUANTUM` ticks, esta tarea es desalojada y su

prioridad pasa a ser baja. Para evitar esta situación, las tareas disponen de la interrupción `0x81`, *ceder*, que indica al sistema operativo que la tarea concede voluntariamente el resto de su quantum a la siguiente tarea. Cuando una tarea con prioridad alta cede su quantum, mantiene su prioridad.

Las tareas con prioridad baja se ejecutan solamente cuando no hay ninguna tarea con prioridad alta, también en el orden en el que están en el arreglo. Cuando una tarea con prioridad baja se ejecuta durante `QUANTUM` ticks, se desaloja y no sucede nada más. Pero si una tarea con prioridad baja cede su quantum dos veces seguidas, su prioridad pasa a ser alta.

Todas las tareas del arreglo que no se están ejecutando, están en estado `LISTA`. La única tarea ejecutándose está en estado `CORRIENDO`. Puede asumir que cuando el sistema arranca, ninguna tarea cedió su quantum.

Las estructuras de datos que se utilizarán para implementar este scheduler son la siguientes:

```
typedef enum {
  ALTA = 0, BAJA = 1
} prioridad_t;

typedef enum {
  CORRIENDO = 0, LISTA = 1
} estado_t;

typedef struct {
  prioridad_t prioridad;
  estado_t estado;
  unsigned short indice_gdt;
  unsigned short cuantas_veces_cedio;
} __attribute__ ((__packed__)) tarea_t;

tarea_t tareas[CANT_TAREAS];
unsigned short indice_tarea_actual;
```

Se pide:

1. (25 puntos) Implementar en lenguaje ensamblador el código correspondiente al scheduler para que se ejecute en la interrupción del timer tick.
2. (20 puntos) Implementar en lenguaje ensamblador el código correspondiente al handler de la interrupción `0x81` *ceder*, en la etiqueta `_isrCeder`.

Aclaraciones:

1. El tamaño del tipo de datos `prioridad_t` y `estado_t` es de 4 bytes.
2. Siempre hay alguna tarea lista para ejecutar.
3. Se puede llamar a una función `proxima_tarea_alta` que devuelve en `ax` el índice de la próxima tarea con prioridad alta, o `CANT_TAREAS + 1` si no hay ninguna.
4. Se puede llamar a una función `proxima_tarea_baja` que devuelve en `ax` el índice de la próxima tarea con prioridad baja. Siempre habrá una tarea con prioridad baja lista para ejecutarse.
5. Tener en cuenta que varias funcionalidades a implementar van a ser utilizadas en ambos ítems de este ejercicio.

Ej. 3. (10 puntos)

Suponiendo que se está utilizando un modelo de segmentación flat con segmentos de 4 GB superpuestos, y se tiene activada paginación, ¿es posible proteger de alguna forma que el usuario modifique el código durante la ejecución del mismo? Justifique.