Examen Febrero de 2013

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida del examen.

Formato

- **Indique su nombre completo y número de cédula en cada hoja (No se corregirán las hojas sin nombre, sin excepciones). Numere todas las hojas e indique la cantidad total de hojas que entrega en la primera.**
- **Escriba las hojas de un solo lado y empiece cada problema en una hoja nueva y cada parte del problema de teórico en una hoja nueva.**
- **Si se entregan varias versiones de un problema solo se corregirá el primero de ellos.**

Dudas

- **Sólo se contestarán dudas de letra.**
- **No se aceptarán dudas en los últimos 30 minutos del examen.**

Material

- **El examen es SIN material (no puede utilizarse ningún apunte, libro ni calculadora). Sólo puede tenerse las hojas del examen, lápiz, goma y lapicera en su banco. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.**

Aprobación

- **Para aprobar el examen se debe tener un mínimo de 60 puntos.**

Finalización

- **El examen dura 4 horas.**

Problema 1 (36 puntos)

1. Describa los sistemas de multiprocesadores simétricos y asimétricos
2. Qué es y cuáles ventajas propone un sistema de tiempo compartido?
3. Qué permite el modo dual de ejecución que brindan ciertos microprocesadores ?
4. Dada las características de los procesadores actuales (multi-cores) qué ventaja presenta un sistema operativo tenga soporte de hilos (threads) a nivel del sistema operativo frente a otro que no lo tenga?
5. De qué se encarga el cargador (*loader*) del sistema operativo?
6. En general los sistemas operativos modernos de propósito general dividen el espacio virtual de los procesos en varios componentes. Mencione estos componentes y su función.
7. Compare un esquema de paginación que utiliza varios niveles de tabla de páginas (digamos 3 niveles) contra un sistema de paginación que utiliza solamente un nivel para la tabla de página.
8. Qué es la TLB, cuál es su función y qué sucede con ella cuando hay un cambio de contexto entre dos procesos distintos?
9. En el subsistema de Entrada/Salida, describa dos métodos que se pueden utilizar para efectuar una E/S.

Problema 2 (32 puntos)

Sea un sistema de archivos simplificado que cuenta con las siguientes estructura de datos:

```
const MAX_DIRS = 4096; const MAX_FILES_ON_DIR = 4096; const MAX_SECTORS_ON_DISK = 16384;
type sector = array [0..511] of byte;
type fs = Array [0..(MAX_DIRS-1)] of dir;
type dir = Record
             dir_entry : array [0..(MAX_FILES_ON_DIR-1)] of file;
             used : boolean;
           end;
type file = Record
                    used : boolean;                    // entrada usada o no
                    file_name : array [0..7] of char; // nombre del archivo
                    file_ext : array [0..2] of char;  // extensión del archivo
                    start : integer;                   // direccion de comienzo
                    type : {ARCHIVO, DIRECTORIO};      // tipo de archivo
                    dir_index : 1..(MAX_DIRS-1);       // indice en fs
                    size : integer;                    // tamaño del archivo en bytes
             end;
type fat = array [0..(MAX_SECTORS_ON_DISK-1)] of -1..(MAX_SECTORS_ON_DISK-1);
type disk = Array [0..(MAX_SECTORS_ON_DISK-1)] of sector;
Var    SA : fs;  // sistema de archivos
       D : disk; // disco
       F : fat;
```

Notas generales:

- Las variables D, SA y F son globales por simplicidad y la información del directorio raíz está en `SA[0]`.
- En la fat el valor 0 representa "fin de archivo" y el -1 "sector libre".
- El campo `dir_index` en file es solo usado cuando el archivo es un directorio.
- Toda operación debe dejar el sistema de archivos en un estado consistente.

Se pide:

1. De una cota para el tamaño máximo en bytes de un archivo. (5pts).
2. Implementar una función que determine cuántos archivos de tipo ARCHIVO hay en el sistema (12pts):

   ```
   Function countFiles(): integer;
   ```

3. Implementar una función que borre un archivo con la siguiente definición (15pts):
   `Function delFile(d_padre: 0..(MAX_DIRS-1); name: Array [0..7] of char; ext: Array [0..2] of char): boolean;`
   Donde `d_padre` es el índice del directorio padre donde está el archivo a borrar,, y `name` y `ext` es el nombre del archivo a borrar. La operación debe retornar si finalizó con éxito o no. Esta operación no es aplicable a directorios (debe retornar error).

Solución:

Parte 1)

Para esta parte consideramos que todos los sectores del disco D pueden ser usados para guardar archivos. Los sectores usados para guardar la FAT y el sistema de archivos no pertenecen a este conjunto.

El archivo podría ocupar toda la FAT. Hay que hacer una excepción con el sector 0 dado que 0 es el indicador de fin de archivo por lo tanto no se puede indicar en la FAT que el siguiente sector de un archivo es el 0. La única forma de usar este sector es que sea el primer sector de un archivo.

Suponiendo entonces que no usamos el sector 0 la cantidad máxima de sectores que podría ocupar el archivo es `MAX_SECTORS_ON_DISK` menos 1. De esa forma, el tamaño máximo de un archivo podría ser:

```
(MAX_SECTORS_ON_DISK - 1) * sizeof(sector)
```

Donde `MAX_SECTORS_ON_DISK = 16384, sizeof(sector) = 512`

Entonces:

```
(MAX_SECTORS_ON_DISK - 1) * sizeof(sector) = (16384 – 1) * 512 = 16383 * 512 = 8388096
```

Parte 2)

```
Function countFiles(): Integer
  var cantFiles: Integer;
      i: Integer;
      j: Integer;
Begin
  cantFiles = 0;
  For i := 0 To  MAX_DIRS-1 Do
    if(SA[i].used) Then
      For j := 0 To (MAX_FILES_ON_DIR-1) Do
        If (SA[i].dir_entry[j].used && SA[i].dir_entry[j].type == ARCHIVO ) Then
          cantFiles = cantFiles + 1;
        End If
      End For
    End If
  End For
  Return(cantFiles);
End
```

Parte 3)

```
Function delFile(d_padre: 0..(MAX_FILES_ON_DIR-1); name: Array [0..7] of char; ext: Array
[0..2] of char): Boolean
var indice: Integer
    actualSector: Integer;
    siguienteSector: Integer;
    encontre: Boolean
```

```
Begin
  encontre := False;
  indice := 0;
  // busco el archivo dentro del directorio.
  While (Not encontre && indice < MAX_FILES_ON_DIR)
    encontre := SA[d_padre].dir_entry[indice].used \
                && SA[d_padre].dir_entry[indice].file_name == name \
                && SA[d_padre].dir_entry[indice].file_ext == ext;
      indice := indice + 1;
  End While
  If(encontre) Then
    indice := indice - 1;
    If (SA[d_padre].dir_entry[indice].type  == ARCHIVO) Then
       actualSector := SA[d_padre].dir_entry[indice].start;
       While (actualSector <> 0) Do
         siguienteSector := FAT[actualSector];
         FAT[actualSector] := -1;
         actualSector := siquienteSector;
       End While
       FAT[actualSector] = -1;
      SA[d_padre].dir_entry[indice].used = False;
    Else
       Return(False); // ERROR: Es un directorio
    End If
  Else
    Return(False); // ERROR: No se encontró el archivo
  End IF
End
```

Problema 3 (32 puntos)

Sea una casa de cambios que dispone de 5 cajas. Los clientes primero se dirigen al mostrador de información que, dependiendo del monto a cambiar, los clasificará en clientes normales y VIP y los dejará pasar al sector de cajas con el siguiente criterio:

- Los clientes normales pasan hasta 5 a la vez al sector de cajas (de forma que siempre haya una caja libre para ellos).
- Los clientes VIP tendrán prioridad frente a los normales y, por seguridad, serán atendidos en forma exclusiva (estando solo uno a la vez en el sector de cajas).
- Dentro de los clientes VIP se hará pasar primero al que desee cambiar el monto mayor.
- No se clasificarán más clientes una vez que haya 4 de cualquier tipo esperando por entrar al sector de cajas.

Además hay un supervisor que podrá modificar el tipo de cambio siempre y cuando no haya clientes en el sector de cajas. Este supervisor tendrá prioridad frente a todos los clientes.

Se desea modelar usando ADA las tareas cliente, mostrador y supervisor.

Se dispone de las siguientes funciones:

• **clasificar(monto: in integer): boolean**	Ejecutada por el mostrador. Retorna true si el cliente es VIP.
• **elegir_monto(): integer**	Ejecutada por el cliente para elegir el monto a cambiar.
• **cambiar(monto: in integer)**	Ejecutada por el cliente para realizar el cambio.
• **nuevo_tipo_cambio()**	Ejecutada por el supervisor para modificar el tipo de cambio.
• **otras_tareas_supervisor()**	Ejecutada por el supervisor para realizar otras tareas distintas a modificar el tipo de cambio.

Nota: No se permite utilizar tareas auxiliares.

Solución:

```
Task Type Cliente
  id:integer;
  tipo: {vip, normal}
begin
  monto := elegir_monto()
  Mostrador.permiso(monto, id, tipo);
  if tipo = vip
      Mostrador.entra_vip[id];
        cambiar(monto);
      Mostrador.sale_vip;
  else
      Mostrador.entra_normal(id);
        cambiar(monto);
      Mostrador.sale_normal;
  endif
end
EndTask

Task Mostrador is
      entry supervisor;
      entry fin_cambio_nuevo_tipo;
      entry permiso(in monto:integer, out id: unteger, out tipo: {vip, normal});
      entry entra_vip[1..4];
      entra_vip;
      entry entra_normal(in id: integer);
      entry sale_normal;
End Task

Task body Mostrador is
var
  tipo: array[1..4] of {libre, normal, vip}
  montos: array[1..4] of integer;
  en_caja, libre, monto_mayor, vip_mayor, i: integer;
begin
  loop
  -- Busco lugar libre
     libre := 0;
     monto_mayor := 0;
     vip_mayor := 0;
     for i = 1 to 4
       if tipo[i] == libre
         libre := i;
       else if tipo[i] == vip and montos[i] >= monto_mayor
         vip_mayor = i;
         monto_mayor = montos[i];
       endif
     endfor

  -- Espero clientes nuevos o supervisor
      select
        when en_caja = 0 =>
          accept supervisor;
          en_caja = 5;
      or
          accept fin_cambio_nuevo_tipo;
          en_caja = 0;
```

```
      or when libre > 0 and supervisor'count = 0 =>
            accept permiso(in monto:integer, out id: integer, out tipo: {vip,
normal})
               id = libre;
               if clasificar(monto)
                 tipo = vip;
                 montos[libre] = monto;
               else
                 tipo = normal;
               endif
               tipo[libre] = tipo;
             end;
      or when vip_mayor > 0 and en_caja = 0 and supervisor'count = 0 =>
         accept entra_vip[vip_mayor];
         en_caja = 5;
         tipo[vip_mayor] = libre;
      or accept sale_vip;
         en_caja = 0;
      or when vip_mayor = 0 and en_caja < 5 and supervisor'count = 0 =>
         accept entra_normal(in id: integer)
           tipo[id] = libre;
         end;
         en_caja = en_caja + 1
      or accept sale_normal;
         en_caja = en_caja - 1
      endselect
  endloop
end
End Task

Task Supervisor
begin
  loop
    Mostrador.supervisor;
    nuevo_tipo_cambio();
    Mostrador.fin_cambio_nuevo_tipo;
  endloop
end
EndTask
```