Segundo Parcial Julio 2012

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida de los puntos del parcial.

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.**
- **Si se entregan varias versiones de un problema solo se corregirá el primero de ellos según el orden de hojas.**

Dudas

- **Sólo se contestarán dudas de letra.**
- **No se aceptarán dudas en los últimos 15 minutos del parcial.**

Material

- **El examen es SIN material (no puede utilizarse ningún apunte, dispositivo móvil, 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.**

Finalización

- **El parcial dura 3 horas.**

Problema 1 (12 puntos)

a. Qué componente se utiliza para realizar en forma más eficiente la traducción de direcciones virtuales a físicas? Describa su funcionamiento.

b. Se desea crear un sistema RAID que tenga el mayor rendimiento posible sin importar la confiabilidad. Qué sistema RAID utilizaría para este caso? Justifique.

c. Qué función tiene el planificador de disco? Describa el planificador SCAN.

d. Describa como funciona el método DMA (*Direct Memory Access*).

Problema 2 (18 puntos) (3, 15)

Un sistema operativo administra sus archivos en disco utilizando el método de asignación indexada directa. A su vez dispone de un mapa de bits (bitmap) para representar los bloques libres u ocupados. El sistema dispone de las siguientes estructuras de datos:

```
const MAX_BLOCKS_ON_DISK = 8192;
const MAX_INODE_ON_DISK = 65536;
type block = array [0..4095] of byte;                    // 4096 bytes
type bitmap = array [0..MAX_BLOCKS_ON_DISK-1] of bit;    // 1024 bytes
type dir_entry = Record
                    name : Array [1..12] of char;    // 12 * 8 bits
                    type : (FILE, DIR);              // 1 bit
                    used : Boolean;                  // 1 bit
                    inode_num : Integer;             // 16 bits
                    perms : array [1..14] of bit;    // 14 bits
             End;
type inode = Record
                    inode_num : Integer;             // 16 bits
                    used : Boolean;                  // 1 bit
                    data : Array [1..5] of Integer;  // 5 * 16 bits
                    tope : 0..5;                     // 3 bits
                    type : (FILE, DIR);              // 1 bit
                    size : Integer;                  // 16 bits
                    reserved : array[1..11] of bit; // 11 bits
             End;
type inode_table = Array [0..MAX_INODE_ON_DISK-1] of inode;
type disk = Array [0..MAX_BLOCKS_ON_DISK-1] of block;
Var
      TI : inode_table;
      B : bitmap;
      D : disk;
```

A su vez, se sabe que el directorio raíz es el inodo número 0, que la tabla de inodos, el bitmap, y el disco son globales.

Por otra parte se dispone de los siguientes procedimientos:

- `Procedure readBlock(d: disk; block_num: 0..MAX_BLOCKS_ON_DISK; Var buff: block, Var ok: Boolean);`
  Lee de disco el bloque pasado como parámetro (carga el parametro de salida buff), En el parámetro `ok` se retorna el éxito de la ejecución de la operación.
- `Procedute writeBlock(d : disk; block_num : 0..MAX_BLOCKS_ON_DISK; buff : block, ok : Boolean);`
  Escribe en el bloque pasado como parámetro la información que se encuentra en el parámetro buff. En el parámetro `ok` se retorna el éxito de la ejecución de la operación.
- `Procedute getInode(cam: array of char; Var nro_inodo: Integer; Var ok: Boolean);`
  Retorna en `nro_inodo` el número de inodo correspondiente al camino absoluto (`cam`). En el parámetro `ok` se retorna el éxito de la ejecución de la operación.
- `Procedure dirname(cam : array of char; Var dir: array of char);`
  Retorna en el parámetro `dir` el directorio que contiene al archivo referenciado en el parámetro `cam`.Ejemplo, `dirname('/home/sistoper/arch.txt') = '/home/sistoper'.`
- `Procedure basename(cam : array of char; Var base: array of char);`
  Retorna en el parámetro base el directorio que contiene al archivo referenciado en el parámetro cam. Ejemplo, `basename('/home/sistoper/arch.txt') = 'arch.txt'.`

Se pide:

a. Determine cuantas entradas puede tiene la tabla de directorios de un inodo de tipo directorio (inodo con type igual a DIR.)

b. Implementar una función (borrarFile) que borra un archivo en el sistema (inodos con type igual a `FILE`). El cabezal de la función es el siguiente:

   ```
   Procedure borrarFile(cam: array of char; Var ok : Boolean);
   ```

   El parámetro `cam` representa el camino absoluto al archivo (ej. '`/home/sistoper/arch.txt`'). En el parámetro `ok` se retorna el éxito en la ejecución de la operación.

Solución:

a)
Se utiliza asignación indexada directa, por lo cual calculando la cantidad de entradas de directorio (dir_entry) que entran en un bloque y observando que cada inodo (ya sea de tipo dir o file pero en nuestro caso nos interesa de tipo dir) tendrá como máximo 5 bloques de datos, podemos calcular la cantidad máxima de entradas que posee la tabla de directorios de un inodo de tipo directorio.

En particular, siendo el tamaño de bloque de 4096B y el tamaño de la estructura dir_entry de 128bits (16B), vemos que en cada bloque entran 4096B/16B = 256 entradas.

Como contamos con 5 bloques (máximo), la cantidad de entradas que puede tener la tabla de directorio de un inodo particular es de 5x256=1280 entradas (la cantidad máxima de inodos es de 65536 asi que es coherente, se podrían utilizar todas las entradas en principio).

b)

```
Procedure borrarFile (cam : array of char; VAR ok: Boolean)
var     dirPadre, nomArch: array of char;
        inodoDir, size, block, entry, inodoArch : Integer;
        encontre : Boolean;
        buff : array [1..256] of dir_entry;
begin
    dirname(cam, dirPadre);
    getInode(dirPadre, inodoDir, ok);

    // No existe el directorio padre o no es un directorio
    if ( not ok ) then begin
        return;
    end

    // El padre no es un directorio
    if(TI[inodoDir].type <> DIR) then begin
        ok := false;
        return;
    end

    // Se busca el archivo dentro de directorio
    size := 0;
    block :=  0;
```

```
    encontre := false;
    basename(cam, nomArch);
    while (size <= TI[inodoDir].size) && (!encontre) then begin
        if ((size mod 4096) = 0) then begin
                entry := 1;
                block := block + 1;
                readBlock(d, TI[inodoDir].data[block], buff, ok);
                if (!ok) then  begin
                    return;
                end
        end

        encontre := ((buff[entry].used = true) && (buff[entry].name =
nomArch));
        If (encontre) then begin
            // se verifica que sea un archivo
            if(buff[entry].type = FILE) then begin
                inodoArch := buff[entry].inode_num;
            end
            else
                ok := false;
                return;
            end
        end
        else
            entry := entry + 1;
            size := size + sizeof(dir_entry);
        end
    end

    if(encontre) then begin
        // se liberan los bloques
        for i := 1 to TI[inodoArch].tope do begin
            B[TI[inodoArch].data[i]] = LIBRE;
        end
        // se borra el inodo de la tabla de inodos y se la tabla de
directorios
        TI[inodoArch].used = false;
```

```
        buff[entry].used := false;
        // sa almacena el cambio realizado en la tabla de directorios
        writeBlock(d, TI[inodoDir].data[block], buff, ok);

        // en caso de error se recupera la estructura
        if(not ok) then  begin
            for i := 1 to TI[inodoArch].tope do begin
                B[TI[inodoArch].data[i]] = OCUPADO;
            end
            TI[inodoArch].used = true;
            buff[entry].used := true;
        end
    end
end
```

Problema 3 (32 puntos)

Se desea modelar en ADA un negocio de venta de tortas a crédito. El mismo tiene como actores a Clientes, dos Vendedores, tres Cocineros y un Autorizador de crédito.

Los clientes solicitan el pedido al Vendedor quien consultará a los tres Cocineros de forma que **simultáneamente** verifiquen si alguno de ellos tiene todos los materiales necesarios para elaborar la torta. Con la primer respuesta afirmativa el Vendedor solicitará al Autorizador de crédito que verifique la disponibilidad de crédito para ese cliente. De tenerla le confirmará al primer cocinero que respondió afirmativamente que realice el pedido y luego le entregará la torta al cliente. Si no hay respuesta afirmativa de ninguno de los Cocineros o si el Autorizador de crédito deniega la solicitud, el Vendedor le responderá al cliente que no es posible elaborar su pedido.

Si el Cocinero no responde **por estar ocupado** cuando el Vendedor le consulta, el Vendedor lo tomará como respuesta negativa. Los cocineros que respondan afirmativamente quedaran esperando a que el Vendedor les indique si les corresponde o no hacer la torta.

Se dispone de las siguientes funciones:

- mi_cedula():Cedula que ejecutado por un Cliente le devuelve número de cédula.
- que_vendedor_soy():{1,2} que ejecutado por un Vendedor le indica su número de vendedor.
- elegir_vendedor():{1,2} que ejecutado por un Cliente le indica el número de vendedor con quien comunicarse.
- elegir_torta():TipoTorta que ejecutado por un Cliente le indica el tipo de torta que desea.
- verificar_materiales(TipoTorta):boolean que ejecutado por un Cocinero retorna si éste tiene los materiales necesarios para elaborar la torta.
- elaborar_torta(TipoTorta):Torta que ejecutado por un Cocinero elabora la torta requerida.
- autorizar_credito(NumeroVendedor, Cedula):boolean que ejecutado por el Autorizador de crédito indica si el Cliente tiene crédito.

Notas:

- Entre el Vendedor y el Autorizador no se puede realizar más de 1 encuentro por consulta.
- Los vendedores no podrán comunicarse entre sí.
- Se podrá utilizar una tarea auxiliar como máximo.
- Cada vendedor puede atender un único cliente a la vez.

Solución 1:

```
TASK TYPE Cliente is
END Cliente;

TASK BODY Cliente is
VAR
  cedula: Cedula;
  tipo: TipoTorta;
  torta: Torta;
BEGIN
  cedula := mi_cedula();
  tipo := elegir_torta();
  vendedores[elegir_vendedor()].atender(cedula, tipo, torta);
  -- si no se puede atender el pedido torta es NIL
END Cliente;

TASK TYPE Vendedor is
  ENTRY atender(cedula: IN Cedula, tipo: IN TipoTorta, torta: OUT Torta);
  ENTRY respuesta_cocinero(c_cedula: IN Cedula, ok: INOUT BOOLEAN);
  ENTRY torta_elaborada(m_torta: IN Torta);
END Vendedor;

TASK BODY Vendedor is
VAR
  i, cant, vendedor: INTEGER;
  m_ok : BOOLEAN;
BEGIN
  LOOP
    vendedor := que_vendedor_soy();
    SELECT
      ACCEPT atender(cedula: IN Cedula, tipo: IN TipoTorta, torta: OUT
Torta)
       torta := NIL;
       cant := 3;
       m_ok := FALSE;
       FOR i := 1 to 3 DO
         SELECT
          cocineros[i].puede_elaborar(vendedor, cedula, tipo);
         ELSE
            cant := cant - 1;
          ENDSELECT;
       ENDFOR
       WHILE cant > 0 DO -- Si aun espero respuesta de cocinero
         ACCEPT respuesta_cocinero(c_cedula: IN Cedula, ok: INOUT
BOOLEAN)
           IF c_cedula = cedula THEN
            IF ok THEN
              cant := 0; -- No atiendo mas cocineros
              Autorizador.autoriza(vendedor, cedula, ok);
              m_ok := ok;
            ELSE
              cant := cant - 1; -- Este cocinero no puede elaborar
            ENDIF
```

```
            ELSE
              ok:= FALSE; -- Descarto elaboracion de un pedido anterior
            ENDIF
          END;
        ENDWHILE;
        IF m_ok THEN -- Si le pedi a un cocinero que elabore la torta
          ACCEPT torta_elaborada(m_torta: IN Torta)
            torta := m_torta;
          END;
        END;
      END;
    OR ACCEPT respuesta_cocinero(c_cedula: IN Cedula, ok: INOUT BOOLEAN)
         ok := FALSE; -- Descarto elaboración de un pedido anterior
     END;
    ENDSELECT;
  ENDLOOP;
END Vendedor;

TASK TYPE Cocinero is
  ENTRY puede_elaborar(vendedor: IN INTEGER, cedula: IN Cedula, tipo: IN
TipoTorta);
END Cocinero;

TASK BODY Cocinero is
VAR
  m_vendedor: INTEGER;
  m_cedula: Cedula;
  m_tipo: TipoTorta;
  ok: BOOLEAN;
  torta: Torta;
BEGIN
  LOOP
    ACCEPT puede_elaborar(vendedor: IN INTEGER, cedula: IN Cedula, tipo:
IN
TipoTorta);
     m_vendedor := vendedor;
     m_cedula := cedula;
     m_tipo := tipo;
    END;
    ok := verificar_materiales(m_tipo);
    Vendedores[m_vendedor].respuesta_cocinero(m_cedula, ok);
    IF ok THEN -- Si debo elaborar torta
      Torta := elaborar_torta(m_tipo);
      Vendedores[m_vendedor].torta_elaborada(torta);
    ENDIF
  ENDLOOP;
End Cocinero;

TASK Autorizador is
  ENTRY autoriza(vendedor: IN INTEGER, cedula: IN Cedula, ok: OUT
BOOLEAN);
End Autorizador;

TASK BODY Autorizador is
BEGIN
```

```
  LOOP
    ACCEPT autoriza(vendedor: IN INTEGER, cedula: IN Cedula, ok: OUT
BOOLEAN)
      ok := autorizar_credito(vendedor, cedula);
    END;
  ENDLOOP;
END Autorizador;

Vendedor vendedores[1..2];
Cocinero cocineros[1..3];


Solución 2:

TASK TYPE Cliente is
END Cliente;
TASK BODY Cliente is
VAR
  cedula: Cedula;
  tipo: TipoTorta;
  torta: Torta;
BEGIN
  cedula := mi_cedula();
  tipo := elegir_torta();
  vendedores[elegir_vendedor()].atender(cedula, tipo, torta);
  -- si no se puede atender el pedido torta es NIL
END Cliente;

TASK TYPE Vendedor is
  ENTRY atender(cedula: IN Cedula, tipo: IN TipoTorta, torta: OUT Torta);
  ENTRY respuesta_cocinero(c_cedula: IN Cedula, ok: INOUT BOOLEAN);
  ENTRY torta_elaborada(m_torta: IN Torta);
END Vendedor;
TASK BODY Vendedor is
VAR
  i, cant, vendedor: INTEGER;
  m_ok : BOOLEAN;
  cocineros_libres: ARRAY [1..3] OF BOOLEAN;
BEGIN
  LOOP
    vendedor := que_vendedor_soy();
    SELECT
      ACCEPT atender(cedula: IN Cedula, tipo: IN TipoTorta, torta: OUT
Torta)
       torta := NIL;
       cant := 0;
       m_ok := FALSE;
       Admin.obtener_cocineros(cocineros_libres);
       FOR i := 1 to 3 DO
         IF cocineros_libres[i] THEN
          cocineros[i].puede_elaborar(vendedor, cedula, tipo);
          cant := cant + 1;
         ENDIF
       ENDFOR
       WHILE cant > 0 DO -- Si aun espero respuesta de cocinero
```

```
          ACCEPT respuesta_cocinero(c_cedula: IN Cedula, ok: INOUT
BOOLEAN)
            IF c_cedula = cedula THEN
              IF ok THEN
                cant := 0; -- No atiendo mas cocineros
                Autorizador.autoriza(vendedor, cedula, ok);
                m_ok := ok;
              ELSE
                cant := cant - 1; -- Este cocinero no puede elaborar
              ENDIF
            ELSE
              ok:= FALSE; -- Descarto elaboracion de un pedido anterior
            ENDIF
          END;
        ENDWHILE;
        IF m_ok THEN -- Si le pedi a un cocinero que elabore la torta
          ACCEPT torta_elaborada(m_torta: IN Torta)
            torta := m_torta;
          END;
        END;
      END;
    OR ACCEPT respuesta_cocinero(c_cedula: IN Cedula, ok: INOUT BOOLEAN)
        ok := FALSE; -- Descarto elaboración de un pedido anterior
      END;
    ENDSELECT;
  ENDLOOP;
END Vendedor;

TASK Admin is
  ENTRY obtener_cocineros(cocineros: OUT ARRAY [1..3] OF INTEGER);
  ENTRY liberar_cocinero(cocinero: IN INTEGER);
  ENTRY id_cocinero(id: OUT INTEGER);
END Admin;
TASK BODY Admin is
VAR
  cocineros_libres: ARRAY [1..3] OF BOOLEAN;
  i, id_cocinero: INTEGER;
BEGIN
  FOR i := 1 TO 3 DO
    cocineros[i].obtener_id(i);
    cocineros_libres[i] := TRUE;
  ENDFOR
  LOOP
    SELECT
       ACCEPT obtener_cocineros(cocineros: OUT ARRAY [1..3] OF INTEGER);
        cocineros := cocineros_libres;
       END
      FOR i := 1 TO 3 DO
        cocineros_libres[i] := FALSE;
      ENDFOR
    OR
       ACCEPT liberar_cocinero(cocinero: IN INTEGER)
        id_cocinero := cocinero;
       END
      cocineros_libres[id_cocinero] := TRUE;
```

```
    ENDSELECT;
END Admin

TASK TYPE Cocinero is
  ENTRY puede_elaborar(vendedor: IN INTEGER, cedula: IN Cedula, tipo: IN
TipoTorta);
  ENTRY obtener_id(idc: IN INTEGER);
END Cocinero;
TASK BODY Cocinero is
VAR
  m_vendedor: INTEGER;
  m_cedula: Cedula;
  m_tipo: TipoTorta;
  ok: BOOLEAN;
  torta: Torta;
  id: INTEGER;
BEGIN
  ACCEPT obtener_id(idc: IN INTEGER)
    id := idc;
  END
  LOOP
    ACCEPT puede_elaborar(vendedor: IN INTEGER, cedula: IN Cedula, tipo:
IN
TipoTorta)
     m_vendedor := vendedor;
     m_cedula := cedula;
     m_tipo := tipo;
    END;
    ok := verificar_materiales(m_tipo);
    Vendedores[m_vendedor].respuesta_cocinero(id, m_cedula, ok);
    IF ok THEN -- Si debo elaborar torta
      Torta := elaborar_torta(m_tipo);
      Vendedores[m_vendedor].torta_elaborada(torta);
    ENDIF
    Admin.liberar_cocinero(id);
  ENDLOOP;
End Cocinero;

TASK Autorizador is
  ENTRY autoriza(vendedor: IN INTEGER, cedula: IN Cedula, ok: OUT
BOOLEAN);
End Autorizador;
TASK BODY Autorizador is
BEGIN
  LOOP
    ACCEPT autoriza(vendedor: IN INTEGER, cedula: IN Cedula, ok: OUT
BOOLEAN)
     ok := autorizar_credito(vendedor, cedula);
    END;
  ENDLOOP;
END Autorizador;

Vendedor vendedores[1..2];
Cocinero cocineros[1..3];
```