Examen Diciembre de 2014

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.**
- **Al momento de finalizar el examen no se podrá escribir absolutamente nada en las hojas, debiéndose parar e ir a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración forma parte de la duración del examen.**

Problema 1 (32 puntos)

1. En el contexto de Memoria Virtual, plantee la ecuación para hallar la EAT (affective access time) y explique cada uno de sus términos.
2.
   i. ¿Qué problemas introduce la caché en sistemas multi-procesadores?
   ii. ¿Qué técnicas se usan para controlarlos?
3. Explique cómo funciona el algoritmo NRU (not recently used).
4. Describa tres métodos de planificación de disco.
5.
   i. Describa tres mecanismos que se pueden utilizar para administrar el espacio libre de memoria.
   ii. Indique qué es la fragmentación interna y la fragmentación externa.
6. Describa cómo funcionan las operaciones signal() y wait() sobre variables de tipo Condition en monitores.
7.
   i. Describa los cuatro registros que generalmente componen un puerto de E/S.
   ii. Explique el método de entrada/salida por interrupciones.
8.
   i. Indique qué función cumple el Despachador.
   ii. Describa los pasos que debe realizar el Despachador para cumplir su tarea.

Problema 2 (35 puntos)

Un sistema de archivos utiliza una estrategia indexada combinada multinivel de dos niveles y un mapa de bits para administrar el espacio libre del disco. En la estructura indexada combinada se dispone de 8 bloques de primer nivel y 1 bloque indirección simple. A continuación se presentan las estructuras de datos utilizadas por este sistema de archivos.

```
const MAX_BLOQUES = 65536;
const MAX_INODOS = 8192;

type bloque = array [0..1023] of byte;              // 1024 bytes
type mapa_bits = array [0..MAX_BLOQUES-1] of bit;

type entrada_dir = Record
      usado : boolean;                          // 1 bit
      nombre : array [0..123] of char;          // 124 bytes
      es_dir : boolean;                         // 1 bit
      inodo_num : int16;                        // 2 bytes
      permisos : array [0..13] of bit;          // 14 bits
End; // 128 bytes

type inodo = Record
      usado : boolean;                          // 1 bit
      inodo_num : int16;                        // 2 bytes
      es_dir : boolean;                         // 1 bit
      tamaño : int32;                           // 4 bytes
      directo : array [0..7] of int16;          // 16 bytes
      directo_tope : int16;                     // 2 bytes
      indirecto : int16;                        // 2 bytes
      indirecto_tope : int16;             // 2 bytes
      reservado : array [0..29] of bit;  // 30 bits
End; // 32 bytes

type inodos_tabla = array [0..MAX_INODOS-1] of inodo;
type disco = array [0..MAX_BLOQUES-1] of bloque;

var
      IT : inodos_tabla;
      MB : mapa_bits;
      D  : disco;
```

Se sabe que:

- las variables `IT`, `MB`, y `D` son globales
- el inodo número 0 es el directorio raíz
- las entradas de los directorios son almacenadas utilizando un array de entradas
- en el mapa de bits, los bloques libres son marcados con valor 1 (*true*)

Se dispone de los siguientes procedimientos:

- `Procedure leerBloque(d : disco; bloque_num : 0..MAX_BLOQUES-1; var buffer : array [0..1023] of byte; var ok : boolean);`

Lee desde el disco `d` el bloque con índice `bloque_num` en la variable `buffer`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

- `Procedure escribirBloque(d : disco; block_num : 0..MAX_BLOQUES-1; buffer : array [0..1023] of bytes; var ok : boolean);`

  Escribe en el bloque con índice `bloque_num` del disco `d` la información que se encuentra en la variable `buffer`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

- `Procedure obtenerInodo(camino : array of char; var inodo_num : integer; var ok : boolean);`

  Retorna en la variable `inodo_num` el número de inodo correspondiente al camino absoluto `camino`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

- `Procedure obtenerNombreDirectorio(camino : array of char; var dir : array of char);`

  Retorna en el parámetro `dir` el nombre del directorio que contiene el archivo referenciado en el parámetro `camino`. Por ejemplo:
  `obtenerNombreDirectorio('/home/sistoper/a.txt') = '/home/sistoper'`

- `Procedure obtenerNombreBase(camino : array of char; var base : array of char);`

  Retorna en el parámetro `base` el nombre del archivo referenciado en el parámetro `camino`. Por ejemplo: `obtenerNombreDirectorio('/home/sistoper/a.txt') = 'a.txt'`

- `Procedure insertarEntradaDir(inodo_dir : 0..MAX_INODOS-1; entrada : entrada_dir; var ok : boolean);`

  Agrega la entrada dada por el parámetro entrada_dir en el directorio dado por el parámetro `inodo_dir`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

Se pide:

1. Indique el tamaño máximo que puede tener un archivo utilizando el sistema de archivos planteado.
2. Implemente una función que encuentre y retorne un bloque libre en el disco. La función a implementar tiene la siguiente firma:

`Procedure buscarBloqueLibre(var bloque : 0..MAX_BLOQUES-1; var ok : boolean);`

   El parámetro `bloque` es el número de bloque libre encontrado. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

3. Implemente una función que dado un directorio, busque una entrada en ese directorio por su nombre. La función a implementar tiene la siguiente firma:

`Procedure buscarEntradaDir(inodo_dir : 0..MAX_INODOS-1; nombre_entrada : array of char; var existe : boolean; var entrada : entrada_dir; var ok : boolean);`

   El parámetro `inodo_dir` representa el número de inodo del directorio donde se realizará la búsqueda y el parámetro `nombre_entrada` es el nombre de entrada buscado (sea archivo o directorio). La función debe retornar verdadero en el parámetro `existe` en caso de encontrar una entrada con el nombre dado y el contenido de la entrada en el parámetro

`entrada`. La función retorna verdadero en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

4. Implemente una función que realice una copia de un archivo en el sistema de archivos. La función a implementar tiene la siguiente firma:

`Procedure copiarArchivo(origen : array of char; destino : array of char; var ok : boolean);`

El parámetro `origen` representa el camino absoluto al archivo original (p.ej.: '`/home/sistoper/a.txt`') y el parámetro `destino` representa el camino absoluto a la nueva copia del archivo (p.ej.: '`/tmp/b.txt`'). La función debe retornar verdadero en el parámetro `ok` para indicar el éxito de la operación. En caso de que ocurra una colisión de nombres con otro archivo o directorio, la función debe fallar.

Para la implementación de la función `copiarArchivo` se puede asumir que:

1. el archivo a copiar es pequeño y nunca utiliza el bloque indirecto
2. en caso de error, no es necesario deshacer los cambios realizados en el sistema de archivos.

Solución:

Parte 1)

```
Cada puntero a bloque ocupa 2 bytes.
Cada inodo puede direccionar como máximo 8 bloques directos a datos más un bloque
indirecto que contiene como máximo 512 punteros a bloques de datos.
  8 + 512 = 520 bloques de datos.
  Tamaño maximo (bloques) = min(MAX_BLOQUES, 520)
  MAX_BLOQUES = 65536

Por lo tanto el tamaño máximo de archivo es 520Kb
```

Parte 2)

```
 Procedure buscarBloqueLibre(var bloque : 0..MAX_BLOQUES-1; var ok : boolean)

 begin
   for bloque = 0 to MAX_BLOQUES-1
     if(MB[bloque] == true)
       ok := true;
       return;
     endif
   endfor
   ok := false;
 end;
```

Parte 3)

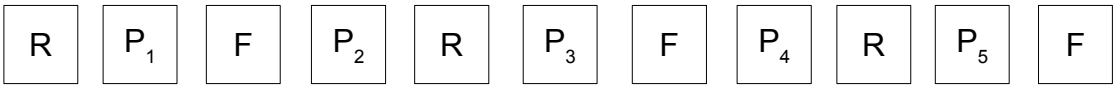
```
  Procedure buscarEntradaDir(inodo_dir : 0..MAX_INODOS-1; nombre_entrada : array
of char; var existe : boolean; var entrada : entrada_dir; var ok : boolean)
  var
    dir : inodo;
    idx,idx2 : int;
    entradas : array[0..8] of entrada_dir;
    punteros : array[0..511] of int16;
  begin
    dir := IT[inodo_dir];
    existe :=false;
    if(dir.usado == false or dir.es_dir == false)
    begin
      ok := false;
      return;
    end;
    ok:= true;

    for(idx := 0; idx < dir.directo_tope; idx++)
      leerBloque(D, dir.directo[idx], entradas, ok);
      if(ok == false)
         return;
      endif;
      for(idx2 := 0; idx2 < 8; idx2 ++)
        if(entradas[idx2].usado == true AND
           entradas[idx2].nombre == nombre_entrada)
             existe := true;
             entrada = entradas[idx2];
             return;
        endif
      endfor
    endfor

    if(dir.indirecto_tope <> 0)
      leerBloque(D, dir.indirecto; punteros, ok);
      if(ok == false)
         return;
      endif;
      for(idx := 0; idx < dir.indirecto_tope; idx++)
        leerBloque(D, punteros[idx], entradas, ok);
        if(ok == false)
           return;
        endif;
        for(idx2 := 0; idx2 < 8; idx2++)
          if(entradas[idx2].usado == true AND
             entradas[idx2].nombre == nombre_entrada)
              existe := true;
              entrada = entradas[idx2];
              return;
          endif
        endfor
      endfor
    endif
  end;
```

Parte 4)

```
Procedure copiarArchivo(origen : arr_char; destino : arr_char; var ok : boolean)
  var
    dir, nombre: array_of_char;
    inodo_origen, inodo_destino : integer;
    entrada : entrada_dir;
    existe : boolean;
    punteros : array[0..511] of int16;
    datos : bloque;
    idx,idx2,libre : integer;
  begin
    obtenerInodo(origen, inodo_origen, ok);
    if(ok == false or inodo_origen.es_dir == true)
      of:=false;
      return;
    endif
    obtenerNombreDirectorio(destino, dir);
    obtenerInodo(dir, inodo_destino, ok);
    if(ok==false)return;

    obtenerNombreBase(destino, nombre);
    buscarEntradaDir(inodo_destino, nombre, existe, entrada, ok);
    if(ok == false or existe == true)
      ok := false;
      return;
    endif

    // Ahora copio bloques de datos
    for(idx = 0; idx < MAX_INODOS-1; idx++)
      if(IT[idx].usado == false)
        entrada.usado = true;
        entrada.nombre = nombre;
        entrada.es_dir = false;
        entrada.inodo_num = idx;
        entrada.permisos = ...;

        insertarEntradaDir(inodo_destino, entrada, ok);
        if(ok==false)return;
        IT[idx].usado = true;

        // directos
        for(idx2 = 0; idx2 < inodo_origen.tope_directo; idx2++)
          buscarBloqueLibre(libre, ok);
          MB[libre] := false;
          if(ok==false)return;
          leerBloque(D, inodo_origen.directo[idx2], datos, ok);
          if(ok==false)return;
          escribirBloque(D, libre, datos, ok);
          if(ok==false)return;
          IT[idx].directo[idx2]=libre;
        endfor
        IT[idx].tope_directo=inodo_origen.tope_directo;
        IT[idx].tope_indirecto=0; // no se usa el bloque indirecto
        IT[idx].tamano=inodo_origen.tamano;
        IT[idx].reservado=inodo_origen.reservado;
        IT[idx].es_dir=inodo_origen.es_dir;
        return;
      endif
    endfor
    ok:= false;
  end;
```

Problema 3 (33 puntos)

Se está realizando una auditoría en una importante empresa. Para ello los auditores se disponen de a pares (hay cinco parejas) a lo largo de una mesa de la siguiente manera:



donde R es una caja de recibos, F es una caja de facturas y P_1 a P_5 son las cinco parejas de auditores. Las parejas deberán ocupar cualquiera de los cinco lugares que se encuentre libre al comenzar a trabajar.

Cada pareja deberá acceder simultáneamente a los recibos y las facturas que tiene a su lado para buscar una factura y su recibo correspondiente. A una misma caja de facturas o recibos no pueden acceder dos parejas a la vez. Luego de terminada la búsqueda las cajas quedan libres. No es aceptable que un auditor se reserve el acceso a una caja de recibos o facturas si no la va a usar inmediatamente.

Una vez encontrada la factura y recibo los auditores deberán analizar el proceso de compra. El resultado del análisis, junto con la factura y la boleta, son enviados al supervisor para su archivo.

Se desea modelar en ADA las tareas pareja y supervisor. Se permitirá como máximo una tarea auxiliar.

Se dispone de las siguientes funciones auxiliares:

- `buscar_par():{factura, recibo}` que, ejecutado por una pareja, busca una factura y su recibo correspondiente en las cajas colindantes a la pareja
- `analizar_compra(factura, recibo): resultado` que, ejecutado por una pareja, realiza el análisis de la compra.
- `archivar(factura, recibo, resultado)` que, ejecutado por el supervisor, archiva los datos.

Solución:

```
task type Pareja is
var
      id_lugar : integer;
      factura : Factura;
      recibo : Recibo;
      resultado : Resultado;
begin
      id_lugar = Mesa.Entro();
      loop
            Mesa.PidoCajas[id_lugar];
            (factura, recibo) = busco_par();
            Mesa.LiberoCajas[id_lugar];
            resultado = analizar_compra(factura, recibo);
            Supervisor.Archivar(factura, recibo, resultado);
      end loop
end

task Mesa is
      entry Entro(out lugar: integer)
      entry PidoCajas[1..5]
      entry LiberoCajas(in lugar: integer)
end task

task body Mesa is
var
      parejas : integer;
      libres : array [1..5] of boolean;
begin
      parejas = 1;
      for caja = 1 to 5
            recibos[libres] = true;
      end
      loop
            select
                  accept Entro(lugar)
                        lugar = parejas
                  end
                  parejas++;
            or
                  when libres[2] =>
                  accept PidoCajas[1];
                  libres[1] = false;
            or
                  when libres[1] and libres[3] =>
                  accept PidoCajas[2];
                  libres[2] = false;
            or
                  when libres[2] and libres[4] =>
                  accept PidoCajas[3];
                  libres[3] = false;
            or
                  when libres[3] and libres[5] =>
                  accept PidoCajas[4];
                  libres[3] = false;
            or
                  when libres[4] =>
                  accept PidoCajas[5];
```

```
                  libres[5] = false;
            or
                  accept LiberoCajas(lugar);
                  libres[lugar] = true;
            end accept
      end loop
end task
task Supervisor is
      entry Archivar
end task

task body Supervisor is
var
      factura : Factura;
      recibo : Recibo;
      resultado : Resultado;
begin
      loop
            Accept Archivar(fac, rec, res)
                  factura = fac;
                  recibo = rec;
                  resultado = res;
            end;
            archivar(factura, recibo, resultado);
      end loop
end
end task

parejas: array [1..5] of Pareja;
```