Segundo Parcial Julio 2017

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 parcial, 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 15 minutos.**
- **Al momento de finalizar el parcial 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 parcial.**

Problema 1 (10 puntos)

1. Describa brevemente los mecanismos de Segmentación y Paginación y compare ventajas y desventajas.
2. Describa los algoritmos de planificación de disco LOOK, C-LOOK y SSTF.
3. En el marco de la seguridad, explique en que contexto se usan las claves asimétricas y como se usan. Mencione alguna ventaja y desventaja respecto a claves simétricas.
4. En el contexto del subsistema de E/S, describa qué es y para qué se usa un ”buffer”.
5. Explique qué es la hiperpaginación y describa un método para detectarlo.

Problema 2 (23 puntos) (7, 8, 8)

Un sistema operativo administra sus archivos en disco utilizando el método de asignación indexada directa, además, permite que un archivo esté en más de un directorio a la vez (enlaces duros/*hard links*). Considere las siguientes estructuras de datos:

```
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
      used : boolean; // 1 bit
      name : array [1..12] of char; // 12 bytes
      type : (file, dir); // 1 bit
      inodo_num : int; // 2 bytes
      reservado : array [1..14] of bit; // 14 bits
End; // 16 bytes

type inodo = Record
      used : boolean; // 1 bit
      inodo_num : int; // 2 bytes
      type : (file, dir); // 1 bit
      size : long; // 4 bytes
      data : array [1..8] of int; // 16 bytes
      tope : int; // 2 bytes
      reference : int; // 2 bytes
      reservado : array [0..45] of bit; // 46 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;  // bit=0 indica bloque libre, bit=1 indica ocupado
D : disco;
```

Notas generales:

- Las variables IT, MB y D son globales.
- El directorio raíz es el inodo número 0.
- El campo reference es utilizado en el inodo para saber cuantas referencias a dicho inodo existen.
- Se cuenta con la función `readBlock(d: disk; block_num: 0..max_blocks_on_disk; Var buff : block);` que lee del disco el bloque pasado por parámetro.
- Se cuenta con la función `writeBlock(d: disk; block_num: 0..max_blocks_on_disk; Var buff : block);` que escribe en disco el bloque pasado por parámetro.

Se pide:

1.
   a) ¿Cómo se podría modificar la estructura para que se pueda disponer de dos bloques de indirección simple? Justifique.
   b) Considerando la adaptación de la parte a, indique la cantidad máxima de entradas que puede contener un directorio. Justifique.

Para las partes 2 y 3 considerar la estructura original.

2. Implementar una función que, dado un inodo, elimina un archivo (no directorio) **por completo** asociado a él (incluyendo todos los enlaces que apuntan a él).
   `Function deleteFile(inodo: Integer): boolean;`
   Donde `inodo` es el número de inodo cuyo archivo se desea eliminar.
3. Implementar una función que agregue un hard link a un archivo dentro de un directorio dado.
   `Function addHardLink(inodo_dir: Integer, inodo_hl: Integer, name: array[1..12] of char): boolean;`
   Donde `inodo_dir` es el número de inodo directorio donde se debe agregar archivo cuyo número de inodo es `inodo_hl`.
   Nota: Suponga que hay al menos una entrada marcada como libre en el directorio dado

SOLUCIÓN:

1.

a)

```
type inodo = Record
      used : boolean; // 1 bit
      inodo_num : int; // 2 bytes
      type : (file, dir); // 1 bit
      size : long; // 4 bytes
      data : array [1..8] of int; // 16 bytes
      tope : int; // 2 bytes
      reference : int; // 2 bytes
      indirecto: array [1..2] of int; //4 bytes
      indirectoTope: int; //2 bytes
      reservado : array [0..45] of bit; // 46 bits
End; // 38 bytes
```

Cuando no hay en uso ningún bloque indirecto, indirectoTope toma el valor 0.
De lo contrario, indica la cantidad de bloques que se esté direccionando.

b)

Cada bloque es de 1024 bytes = 2^{10} bytes

Cada entrada directorio ocupa 16 bytes = 2^4 bytes

→ hay 2^{10} / 2^4 entradas de directorio por bloque = 2^6 entradas por bloque

Se tiene:

- 8 bloques de primer nivel.

- 2 bloque de indirección simple:

  Por cada bloque de indirección simple: 2^{10} bytes por bloque / 2 bytes (necesario para referenciar un bloque) = 2^9 bloques, entonces con un bloque de indirección simple es posible direccionar 512 bloques.

→ Se tiene 2^3 + 2 * 2^9 = 8 + 1024 = 1032 bloques.

Como por bloque se tiene 2^6 entradas, entonces se tiene en total (2^{10} + 2^3)bloques * 2^6 entradas por bloque = (2^{16} + 2^9) entradas de directorio = 66048 entradas de directorio

2.
Solución con recorrida no eficiente del filesystem:

```
function deleteFile(inodo: Integer): boolean;
var cambiado, encontre: boolean; nroInodo, block, entry: Integer;
     buff: array[1..64] of dir_entry;
begin
     if(IT[inodo].used && IT[inodo].type == 'file')then
          nroInodo := 0;
          while(nroInodo < Max_Inodos && IT[inodo].reference > 0) do
               if(IT[nroInodo].used && IT[nroInodo].type == 'dir' &&
               IT[nroInodo].tope > 0) then
                    block := 1;
                    while(block < IT[nroInodo].tope &&
                    IT[inodo].reference > 0) do
                      cambiado := false;
                      read(D, IT[nroInodo].data[block], buff);
                      entry := 1;
                      while (entry <= 64) do
                         if(buff[entry].used and
                         buff[entry].inode_num == inodo) then
                           buff[entry].used := false;
                           IT[inodo].reference--;
                           cambiado := true;
                         end if;
                         entry++;
                      end while;

                      if(cambiado)then
                         write(D, IT[nroInodo].data[block], buff);
                      block++;
                    end while;
               end if
               nroInodo++;
          end while;

          for(int i = 0; i < IT[inodo].tope; i++)
               MB[IT[inodo].data[i]] = 0; // libero bloques
          IT[inodo].used := false;

          return (IT[inodo].reference==0);
     else
          return false;
     end if
end functon;
```

Solución con recorrida eficiente del filesystem:

```
function deleteFile(inodo: Integer): boolean;
var cambiado, encontre: boolean;
     nroInodo, block, entry, tope_pila: Integer;
     buff: array[1..64] of dir_entry;
     pila: array[1..MAX_INODOS] of Integer;
begin
     pila[1] := 0;
     tope_pila := 2;

     if(!IT[inodo].used || IT[inodo].type!='file') return false;

     while (tope_pila > 1 && IT[inodo].reference > 0)
          nroInodo := pila[tope_pila];
          tope_pila--;

          if(IT[nroInodo].used && IT[nroInodo].type == 'dir' &&
          IT[nroInodo].tope > 0) then
               block := 1;
               while(block < IT[nroInodo].tope &&
               IT[inodo].reference > 0) do
                 cambiado := false;
                 read(D, IT[nroInodo].data[block], buff);
                 entry := 1;
                 while (entry <= 64) do
                    if(buff[entry].used &&
                    buff[entry].inode_num == inodo) then
                      buff[entry].used := false;
                      cambiado := true;
                      IT[inodo].reference--;
                    else if(buff[entry].used &&
                    buff[entry].dir == 'dir') then
                      pila[tope_pila] := buff[entry].inode_num;
                      tope_pila++;
                    end if;
                    entry++;
                 end while;

                 if(cambiado)then
                    write(D, IT[nroInodo].data[block], buff);
                 block++;
               end while;
          end if;
     end while;

     for(int i = 0; i < IT[inodo].tope; i++)
          MB[IT[inodo].data[i]] = 0; // libero bloques
     IT[inodo].used := false;

     return (IT[inodo].reference==0);
end functon;
```

3.

```
function addHardLink(inodo_dir: Integer, inodo_hl: Integer, name:array
[1..12] of char): boolean;
var entry_block, entry_id, block, entry, idBlock, i: Integer;
     buff: array[1..8][1..64] of dir_entry
begin
     if(!IT[inodo_dir].used or IT[inodo_dir].type != 'dir' or
          !IT[inodo_hl].used) then return false;
//chequeo que no haya un archivo con el mismo nombre y busco entry libre
     block := 1; entry_block=0;
     while(block <= IT[inodo_dir].tope)do
          read(D, IT[inodo_dir].data[block], buff[block]);
          entry := 1;
          while(entry <= 64)do
               if(entry_block==0 and not(buff[block][entry].used))then
                    entry_block = block;
                    entry_id = entry;
               end if;
               if(buff[block][entry].used and
                    buff[block][entry].name == name)then
                         return false; // ya existe archivo
               end if;
               entry++;
          end while;
          block++;
     end while;

     if(entry_block == 0)
          return false; // No deberia pasar segun la letra

     buff[entry_block][entry_id].used := true;
     buff[entry_block][entry_id].name := name;
     buff[entry_block][entry_id].type := IT[inodo_hl].type;
     buff[entry_block][entry_id].inodo_num := IT[inodo_hl].inodo_num
     write(D, IT[inodo_dir].data[entry_block], buff[entry_block]);
     IT[inodo_hl].reference++;
     return true;
end function;
```

Problema 3 (30 puntos)

Se desea modelar un sistema de transporte de minibuses en el aeropuerto de una gran ciudad. Cada minibus pertenece a uno de los 70 hoteles de la ciudad (puede haber más de un minibus por hotel) y solo viaja del aeropuerto hacia ese hotel. Los minibuses tienen capacidad para 12 personas y solo viajan al hotel cuando están llenos (excepto por razones de seguridad).

Por razones de organización los pasajeros no pueden esperar dentro del minibus y solo podrán subir cuando el minibus esté habilitado para viajar.

Si pasan más de 15 minutos sin que lleguen minibuses ni pasajeros es considerado un problema de seguridad y en ese caso suben las personas que tienen minibuses disponibles y todos los minibuses se van del aeropuerto. Las personas que no tienen minibus continúan esperando. Luego de que se van los minibuses se vuelve nuevamente al funcionamiento normal del aeropuerto.

Se pide: Implementar en ADA las tareas Minibus y Pasajero. Se podrán usar como máximo cinco tareas auxiliares contando cada instancia de tarea.

Se dispone de las siguientes funciones auxiliares:

- queHotel(): [1..70] invocada por el pasajero y por el minibus para saber a que hotel va
- ubicarse_en_bus() invocada por el pasajero para guardar las valijas, subir al minibus y sentarse
- salir() invocada por el minibus para partir hacia el hotel

SOLUCIÓN:

```
Task Type Pasajero is
End Pasajero;

Task body Pasajero is
var hotel:integer;
begin
  hotel = queHotel();
  Aeropuerto.pasajero(hotel);
  Despachador.puedoSubir[hotel]; -- Pido permiso para subir
  ubicarse_en_bus();
  Despachador.yaSubi(hotel);  -- Aviso que subi
End Pasajero;

Task Type Minibus is
End Minibus;

Task body Minibus is
var hotel:integer;
begin
  hotel = queHotel();
  Aeropuerto.minibus(hotel);
  Despachador.puedoSalir[hotel]; -- Pido permiso para salir
  salir();
End Minibus;

Task Aeropuerto is
  entry pasajero(hotel: in integer);
  entry minibus(hotel: in integer);
End Aeropuerto;

Task body Aeropuerto is
var pax, buses: array[1..70] of integer;
var hotel: integer;
begin
  loop
    select
        accept pasajero(m_hotel)  -- Llega un pasajero
          hotel = m_hotel;
        end;
        pax[hotel]++;
        if pax[hotel] = 12 and buses[hotel] > 0  -- Si llego el 12vo pasajero y
          Despachador.despachar(hotel, 12);      -- hay bus, mando a despachar
          pax[hotel]=0;
          buses[hotel]=buses[hotel]-1;
        endif;
     or accept minibus(m_hotel)   -- Llega un minibus
          hotel = m_hotel;
        end;
        if pax[hotel] > 11                       -- Si hay al menos 12 pasajeros
          Despachador.despachar(hotel, 12);      -- mando despachar
          pax[hotel]=pax[hotel]-12;
        else
          buses[hotel]++;
        endif;
     or delay 900  -- Pasaron 15 minutos sin que llegue pasajero o minibus
        for hotel=1 to 70
          loop while buses[hotel] > 0
            Despachador.despachar(hotel, pax[hotel]); -- Despacho minibus
            pax[hotel]=0;                             -- con los pasajeros que
            buses[hotel]=buses[hotel]-1;              -- haya
          endloop;
```

```
        endfor
    endselect;
  endloop;
End Aeropuerto;

Task Despachador is
  entry puedoSubir[1..70]();
  entry puedoSalir[1..70]();
  entry yaSubi(hotel: in integer);
  entry despachar(in integer:hotel, in integer:pax);
End Despachador;

Task body Despachador is
var hotel, pax:integer;
var espero: array[1..70] of integer;
begin
  loop
    select
        accept despachar(m_hotel, m_pax)
          hotel = m_hotel;
          pax = m_pax;
        end;
        loop while espero[hotel] > 0     -- Si habia una emergencia previa
          accept yaSubi[hotel]()         -- para la cual aun no fue totalmente
          espero[hotel]=espero[hotel]-1; -- despachada espero aqui se se despache
          if espero[hotel]=0
            accept puedoSalir[hotel](); -- Dejo salir al minibus
          endif
        endloop
        if pax = 0
          accept puedoSalir[hotel]();    -- Si el despacho es vacio, dejo salir
        else
          loop while pax > 0             -- Sino permito a los pasajeros que
            accept puedoSubir[hotel];    -- se ubiquen en el bus
            espero[hotel]=espero[hotel]+1;
            pax=pax-1;
          endloop;
        endif
     or accept yaSubi(m_hotel)
            hotel = m_hotel;
         end;
         espero[hotel] = espero[hotel]-1;
         if espero[hotel] = 0            -- Cuando se ubica el ultimo pasajero
           accept puedoSalir[hotel]();   -- le permito al bus que se vaya
         endif
    endselect
  endloop;
End Despachador;
```