Examen 19 de diciembre de 2016

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.**
- **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. Compare los métodos de E/S programada y por interrupciones. Indique ventajas y desventajas de cada método.

2. Describa y compare los diseños de sistemas operativos monolítico y en capas.

3. ¿Qué entiende por planificador no expropiativo (non-preemptive scheduler)? ¿Cuál es su principal desventaja respecto a un planificador expropiativo?

4. Ilustre y explique brevemente como se implementa la correspondencia entre direcciones virtuales y físicas en un esquema de segmentación.

5.
   I. Describa la técnica de segmentación de memoria.
   II. ¿Qué ventajas brinda esta técnica con respecto a la técnica de paginación?

6.
   I. ¿Cuándo son ejecutados los algoritmos de reemplazo de marcos (frames)?
   II. Describa el algoritmo de reemplazo LRU (Least Recently Used).

7. Describa el método de asignación FAT (File Allocation Table) en un sistema de archivos.

8.
   I. ¿Qué ventajas brinda los sistema RAID (Redundant Array of Inexpensive Disk)?
   II. Describa el nivel 1 de RAID. Indique un escenario donde su uso es adecuado y otro donde no lo sea.

Problema 2 (34 puntos)

Se desea modelar usando ADA una intersección de una calle de un solo sentido y una peatonal. Por la calle de un solo sentido circulan autos y bicicletas. No se debe permitir el cruce simultáneo de peatones con autos y/o bicicletas. Los autos y las bicicletas pueden cruzar simultáneamente dado que van en el mismo sentido.

Los peatones tienen prioridad sobre autos y bicicletas pero las bicicletas no deben esperar más de 5 minutos cuando están en el borde del cruce. Cuando las bicicletas no pueden pasar por el cruce se forma una fila desde el borde del mismo. Si la espera de la bicicleta que está en el borde del cruce supera los 5 minutos, las bicicletas pasarán a tener prioridad sobre los peatones. Cuando las bicicletas comienzan a pasar se dejará cruzar a todas las que haya en espera con un máximo de 10. Luego de ocurrido esto los peatones volverán a tener prioridad.

Se pide:

Implementar en ADA las tareas Auto, Bicicleta y Peatón. Se permite usar tareas auxiliares.

Se dispone de la siguiente función auxiliar:

- cruzar() Ejecutada por los autos, las bicicletas y los peatones para cruzar la calle.

Solución:

```
Task AdminBici is
  entry biciEspera();
  entry finEspera();
End AdminBici

Task Body AdminBici is
repite: bool;
begin
 loop
   do
    repite = false
    select
        accept biciEspera;
      or accept finEspera;  -- caso de borde si se "cruzan" finEspera
         repite = true;     -- con prioridadBici
      end
   while repite = true;
   select
     accept finEspera
     or delay 5*60
       AdminCruce.prioridadBici;
   endselect
 endloop
End AdminBici

Task AdminCruce is
  entry inicioAuto();
  entry finAuto();
  entry llegaBici(out bool pasar);
  entry inicioBici();
  entry finBici();
  entry inicioPeaton();
  entry finPeaton();
  entry prioridadBici();
End AdminCruce

Task Body AdminCruce is
  bicis, peatones, autos, numbicis: integer;
  prioBici, b_espera: boolean;
begin
  prioBici = false; b_espera = false; bicis=0; peatones=0; autos=0; numbicis=0;
  loop
    select
      when bicis = 0 and autos = 0 and (prioBici = false or llegaBici'count = 0)
           and prioridadBici'count = 0 =>
        accept inicioPeaton;
        peatones = peatones + 1;
        prioBici = false;
      or when peatones = 0 and (prioBici = true or inicioPeaton'count = 0) =>
        accept inicioAuto;
        autos = autos + 1;
      or when prioBici = false and peatones = 0 and inicioPeaton'count = 0 =>
         accept llegaBici(out bool pasar)  -- Llega bici y cruce esta libre
           pasar = true;
         end;
         bicis = bicis + 1;
      or when b_espera = false and bicis = 0 and prioBici = false and
              (peatones > 0 or inicioPeaton'count > 0) =>
         accept llegaBici(out bool pasar)  -- Primer bici espera en borde del cruce
           pasar = false;
         end;
         AdminBici.biciEspera();
         numBicis = 1;
         b_espera = true;
      or when prioBici = true =>
         accept llegaBici(out bool pasar)  -- Pasan hasta 10 bicis
```

```
          pasar = true;
        end;
       bicis = bicis + 1;
       numbicis = numbicis + 1;
       if numbicis = 10
         prioBici = false;
       endif
     or when inicioPeaton'count = 0 or prioBici = true =>
        accept finPeaton;
        peatones = peatones - 1;
        if peatones = 0 and b_espera
          b_espera = false;
          AdminBici.finEspera(); -- Bici ya no esta esperando
          accept inicioBici;  -- Dejo pasar bici en borde del cruce
          bicis = 1
        endif
     or accept finAuto;
       autos = autos - 1;
     or accept finBici;
       bicis = bicis - 1;
     or accept prioridadBici;
       if bicis = 0
         prioBici = true;
         if peatones = 0
           accept inicioBici; -- Dejo pasar bici en borde del cruce
           b_espera = false;
           bicis = 1;
         endif
       endif
   end;
  endloop;
End Admin Cruce

Task Type Auto is
End Auto

Task Body Auto is
  AdminCruce.inicioAuto();
  cruzar();
  AdminCruce.finAuto();
End Auto

Task Type Bicicleta is
End Bicicleta

Task Body Bicicleta is
  pasar:bool;
begin
  AdminCruce.llegaBici(out pasar);
  if not pasar
      AdminCruce.inicioBici();
  end;
  cruzar();
  AdminCruce.finBici();
End Bicicleta

Task Type Peaton is
End Peaton

Task Body Peaton is
  AdminCruce.inicioPeaton();
  cruzar();
  AdminCruce.finPeaton();
End Peaton
```

Problema 3 (34 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 de 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 : int;                     // 2 bytes
      permisos : array [0..13] of bit;          // 14 bits
End; // 128 bytes

type inodo = Record
      usado : boolean;                          // 1 bit
      inodo_num : int;                          // 2 bytes
      es_dir : boolean;                    // 1 bit
      tamaño : long;                            // 4 bytes
      directo : array [0..7] of int;            // 16 bytes
      directo_tope : int;                       // 2 bytes
      indirecto : int;                          // 2 bytes
      indirecto_tope : int;                // 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*).
- Cuándo el bloque indirecto no está en uso, su tope vale 0.
- La estructura de inodo debe ocupar siempre la mínima cantidad de espacio en disco posible.

Se dispone de los siguientes procedimientos:

- `Function leer(d: disco; bloque_num: 0..MAX_BLOQUES-1; var buffer: array [0..1023] of byte) : boolean;`
  Lee desde el disco `d` el bloque con índice `bloque_num` en la variable `buffer`. La función retorna verdadero en caso de que la operación haya sido ejecutada con éxito.

- `Function escribir(d: disco; block_num: 0..MAX_BLOQUES-1; buffer: array [0..1023] of bytes) : 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 caso de que la operación haya sido ejecutada con éxito.

- `Function parteCamino(camino: array of char; var primero: array of char) : array of char;`
  El parámetro `camino` contiene el camino completo de un elemento. La función seteará el nombre del primer elemento del camino en el parámetro `primero`, y retornará el camino con el resto de los elementos. Por ejemplo:

  si `camino='/home/so/a.txt'`, entonces `primero='home'` y se retorna `'so/a.txt'`
  si `camino='a.txt'`, entonces `primero='a.txt'` y se retorna `''`
  si `camino=''`, entonces `primero=''` y se retorna `''`

Se pide:

1. Indique la cantidad máxima de entradas que puede contener un directorio.

2. Indique la cantidad de bloques que ocupa la estructura de un directorio totalmente lleno.

3. Implemente una función que retorne el número de inodo de un elemento determinado por un camino.

   `Function buscarInodo(c: array of char; var inodo_num: int) : boolean;`

   El parámetro `c` es el camino al elemento buscado. La función retorna el número de inodo del elemento buscado en la variable `inodo_num` y retorna `true` en caso de que la operación haya sido ejecutada con éxito.

4. Implemente una función que trunque a 8 bloques el tamaño de un archivo. En el caso de que ocupe más de 8 bloques, se le deberán quitar bloques hasta dejarlo solamente con 8. En el caso de que el archivo ocupe menos de 8 bloques, no se le realizará ninguna acción.

   `Function trunc8(c: array of char) : boolean;`

   El parámetro `c` es el camino al elemento buscado. La función retorna `true` en la variable `ok` en caso de que la operación haya sido ejecutada con éxito.

Nota: Se pueden utilizar (y reutilizar) funciones auxiliares siempre y cuando se encuentren totalmente implementadas.

Solución

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

Cada entrada directorio ocupa 128 bytes = 2^7 bytes

⇨ $2^{10} / 2^7 = 2^3$ entradas directorio por bloque

Por otro lado se tiene:

- 8 bloques de primer nivel.
- 1 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 $2^3 + 2^9 = 8 + 512 = 520$ bloques.

⇨ La cantidad máxima de entradas es: 520 bloques * 2^3 entradas por bloque = 4160 entradas

2) Por parte 1, una estructura referencia a 520 bloques pero además hay que sumarle el bloque de indirección. Por lo tanto ocupa: 520 + 1 = 521 bloques

3)

```
Function buscarInodo(c: array of char; var inodo_num: int) : boolean {
    entrada_dir bloque[8],
    int bloque_ind[512];

    bool ok = true;
    inodo in = IT[0];

    string camino, nombre, resto;
    camino = parteCamino(c, nombre);

    int entry_idx = -1;

    while (nombre != "") {
        for (int i=0; i<in.directo_tope && entry_idx<0; i++) {
            ok = leer(D, in.directo[i], bloque);
            if (!ok) return false;
            entry_idx = buscarEntradaDir(bloque,nombre);
        }

        if (entry_idx < 0 && in.indirecto_tope > 0) {
            ok = leer(D, in.indirecto, bloque_ind);
            if (!ok) return false;

            for (int i=0; i<in.indirecto_tope && entry_idx<0; i++) {
                ok = leer(D, bloque_ind[i], bloque);
                if (!ok) return false;
                entry_idx = buscarEntradaDir(bloque,nombre);
            }
        }

        if (entry_idx >= 0) {
            if (!bloque[entry_idx].es_dir && resto!="") {
                return false; // error, encontró archivo que debía ser directorio
      } else {
                in = IT[bloque[entry_idx].inodo_num];
                camino = parteCamino(camino, nombre);
```

```
                entry_idx = -1;
      }
        } else {
            return false; // error, no encontró
        }
    } // end while

    inodo_num = in.inodo_num;
    return true;
}

int buscarEntradaDir(entrada_dir  bloque, string nombre) {
    for (int i=0; i<8; i++) {
        if (bloque[i].usado && bloque[i].nombre == nombre) {
            return i;
        }
    }
    return -1;
}

4)
Function trunc8(c: array of char) : boolean {
    bool ok = true;
    int bloque_ind[512];

    int idx;
    if (!buscarInodo(c, idx)) return false;

    inode in = IT[idx];
    if (in.es_dir) return false;

    if (in.indirecto_tope > 0) {
        ok = leer(D, in.indirecto, bloque_ind);
       if (!ok) return false;

        for (int i=0; i<inodo.indirecto_tope; i++) {
            MB[bloque_ind[i]] = 1;
        }
        MB[in.indirecto] = 1;
        in.tamaño = 1024 * 8;
        in.indirecto_tope = 0;
    }
}
```