Examen Julio de 2010

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. (No se corregirá la hoja que tenga el ejercicio compartido, sin excepciones).**
- **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 60 puntos.**

Finalización

- **El examen dura 3 horas.**

Problema 1 (60 puntos)

Teórico (25 pts)

Responda las siguientes preguntas justificando cada respuesta:

1.
   a. ¿Qué tarea realiza el planificador de CPU en un sistema operativo?
   b. Defina dos métricas utilizadas para comprar planificadores de CPU.
   c. Realice un diagrama y describa el planificador *Multi-level feedback queue.*
2. ¿Qué tarea tiene asignada el cargador (*loader*) del sistema operativo?
3. Describa la estructura de tablas de página jerárquica (multinivel). Mencione sus ventajas y desventajas. A su vez, describa que componentes permiten mejorar las desventajas.

Práctico (35 pts)

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;
type block = array [0..511] of byte;             // 512 bytes
     bitmap = array [0..8191] 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] of inode;
type disk = Array [0..max_blocks_on_disk] 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. Conjuntamente, cada bloque de datos de los directorios tiene 32 entradas de tipo `dir_entry` y el tamaño (`size`) de un archivo de tipo directorio es el campo tope por el tamaño de bloque.

Se pide:

- ¿Cuántos archivos puede contener un directorio como máximo?
- Implementar una función (`createFile`) que cree un archivo vacío (`size = 0`) en el sistema. El cabezal de la función es el siguiente:

```
Procedure createFile(cam : array of char; t: (file,dir); p: array [1..14]
of bit; Var ok : Boolean);
```

El parámetro `cam` representa el camino absoluto al archivo (ej. '`/home/sistoper/archivo.txt`'). El parámetro `t` y `p` son el tipo (archivo o directorio) y los permisos respectivamente. En el parámetro `ok` se retorna el éxito en la ejecución de la operación.

Nota:

- Defina el cabezal de las operaciones que requiera necesarias para operar sobre el mapa de bits y lectura/escritura de un bloque en disco. No las implemente.
- Asuma que se dispone de las siguientes funciones:
  - `Procedure 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/archivo.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/archivo.txt') = 'archivo.txt'`.

Solución

(Práctica)

a.

Es asignación indexada directa, por lo que cada archivo (file o dir) tendrá como máximo 5 bloques de datos. Como en cada bloque de datos tiene 32 entradas dir_entry, tendremos un máximo de 160 entradas. Por lo que tendremos un máximo de 160 archivos en un directorio.

b.

Pseudocódigo:

Comienzo

Busco un inodo libre en la tabla de inodos, sino error.

Realizar control de que el padre exista y sea un directorio, sino error.

Recorrer todos los datos del directorio padre buscando una entrada libre y que el nombre no se repita. Si tengo que agregar algún bloque más lo hago y dejo todo consistente, sino error.

Conseguí un inodo y el dir_entry libres, por lo que…

Seteo el inodo como ocupado en la tabla y meto la info.

Seteo la info dir_entry en el bloque de datos.

Fin

```
Procedure createFile(cam : array of char; t: (file,dir); p: array [1..14] of bit; Var ok : Boolean);
Var new_inode, dir_inode : 0..max_inode_on_disk;
   Block : block;
   dir, namefile : array of char;
   dentry_pointer, libre : ^dir_entry;
   entry : integer;
Begin
  If (ok = ((new_inode = getFreeInode() == -1))) then
    return;
  fi
  dirname(cam,dir);
  basename(cam,namefile);
  dir_inode = getInode(dir);
  if (ok = (!TI[dir_inode].used or TI[dir_inode].type = file)) then
    setFreeInode(TI,new_inode);
    return;
  fi
  i = 1; entry = 0; libre = NULL;
  while (ok and i <= TI[dir_inode].tope) do
          if (entry == 0) then
                readBlock(D, TI[dir_inode].data[i],block);
                dentry_pointer = (dir_entry *) &(block);
                i++;
```

```
            fi
            if (dentry_pointer->used) then
                  if not (ok = (dentry_pointer->name != nameFile)) then
                        setFreeInode(new_inode);
                        return;
                  fi
            else
                  libre = dentry_pointer; // Al asignar siempre hace que
me quede siempre con el último libre del último bloque que recorra.
            Fi
            entry++;
            dentry_pointer++;
            if (entry == 32)
                  entry = 0;
  fi
  if (libre == null && top == 5) return;
  if (libre == null) and (TI[dir_inode].tope != 5) then
    if ( ok = (new_block = getFreeBlockBITMAP(B)) != -1) then
      setFreeInode(new_inode);
      return;
    fi
    TI[dir_inode].tope++
    TI[dir_inode].data[TI[dir_inode].tope] = new_block;
    TI[dir_inode].size += 512;
    readBlock(D, new_block, block);
    initDirBlock(block);
    libre = (dir_entry *)&block;
  fi
  // Tengo todo, modifico el inodo, el dir_entry y grabo a disco.
  TI[new_inode].inode_num = new_inode,
  TI[new_inode].used = true;
  TI[new_inode].tope = 0;
  TI[new_inode].type = t;
  TI[new_inode].size = 0;
  libre->name = namefile;
  libre->type = t;
  libre->inode_num = new_inode;
  libre->perms = p;
```

```
  libre->used = true;
  writeBlock(D,block);
  return;
End;
Procedure getFreeInode(): integer;
Var
  encontre : boolean;
  i : integer;
Begin
  encontre = false; i = 0;
  // El inodo 0 lo salteo porque es el root.
  while (not encontre) and (i < max_inode_on_disk) do begin
    i++;
    encontre = not (TI[i].used);
  end
  if (encontre) then
    TI[i].used = true;
    return i;
  else
    return -1;
  fi
End;
Procedure setFreeInode(i : 0..max_inode_on_disk);
Begin
  TI[i].used = false;
End;
Procedure initDirBlock(Var b : block);
Var
   dentry_pointer : ^dir_entry;
Begin
  dentry_pointer = &block;
  For (i = 0; i < 32; i++) do begin
    dentry_pointer->used = false;
    dentry_pointer++;
  End;
End;
```

Problema 2 (40 puntos)

Se desea modelar la venta de entradas de un estadio de fútbol mundialista usando monitores. El estadio dispone de 3 boleterías en las que se formarán tres colas con los hinchas que desean comprar entradas (una cola por boletería).

Además hay revendedores (uno por boletería) los cuales tratarán de convencer a los hinchas de que dejen la cola y les compren a ellos sus entradas. Estos revendedores solo intentarán vender sus entradas a los primeros diez hinchas de cada cola. Si el hincha decide comprarle la entrada al revendedor se retirará de la fila en forma inmediata dejando su lugar a la siguiente persona. Lo mismo sucederá si el hincha está hablando con el revendedor cuando avanza la fila. En este caso, si el hincha no decide comprarle al revendedor deberá ingresar nuevamente al final de la cola.

Notas:

- No se permitirán colados en la fila
- No se permitirán tareas auxiliares
- Cada revendedor podrá intentar vender sus entradas en cualquier momento

Se dispone de los siguientes procedimientos auxiliares:

- elegirCola() : [1..3] Ejecutada por el cliente para seleccionar la cola donde va a esperar
- elegirCliente() : [1..10] Ejecutada por los revendedores para seleccionar un cliente interesado en la reventa
- evaluarCompra(): boolean Ejecutada por el cliente para decidir si le compra al revendedor o no
- comprarAlRevendedor() Ejecutada por el cliente para comprar la entrada al revendedor
- comprarEnBoleteria() Ejecutada por el cliente para comprar la entrada en la boletería

==========

```
Cliente() {
      int nro = elegirCola();
      boolean vengoDeReventa = false;
      do {
            boolean boleteria = colas[nro].entrarCola(vengoDeReventa);
            if(boleteria) {
                  comprarEnBoleteria();
                  colas[nro].salir();
            }
            else  {
```

```
                  if(evaluarCompra()) {
                        comprarAlRevendedor();
                        colas[nro].salirRevendedor();
                  }
                  else vengoDeReventa = true;
            }
      } while(!boleteria);
}

Revendedor(int nro) {
      int cli;
      while(true) {
            cli = elegirCliente();
            colas[nro].reventa(cli);
      }
}


type Monitor MonitorCola is
VAR
   condition cola;
   condition revendedor;
   condition espera[1..10];
   int cant := 0;
   int pos_revendedor := -1;

boolean entrarCola(boolean vengoDeReventa)
{
      int pos;
      // Viene de reventa y la cola no se movió
      if(vengoDeReventa && pos_revendedor != -1) {
            pos = pos_revendedor;
            pos_revendedor = -1;
            revendedor.signal();
      }
      else {
            cant++;
            if(cant > 10)
                  pos := 11;
            else
```

```
                  pos := cant;
      }
      while(pos != 1) {
            if(pos == 11)
                  cola.wait();
            else
                  espera[pos].wait();
            if(pos_revendedor == pos)
                  return false;
            int npos = pos + 1;
            if(pos_revendedor == npos) npos++; // Salteo cliente con revendedor
            // Despierto al siguiente
            if(npos > 10)
                  cola.signal;
            else
                  espera[npos].signal();  // Despierto al siguiente
            pos--; // Avanzo 1 lugar en la cola
            if(pos_revendedor == pos) {
                  pos_revendedor = -1;
                  pos--; // Me muevo 2 lugares
            }
      }
      return true;
}

void salirRevendedor()
{
      cant--;
      int pos = pos_revendedor;
      pos_revendedor = -1;
      if(pos == 10)
            cola.signal
      else
            if (pos != -1)
                  espera[pos+1].signal();
      revendedor.signal();
}

void salir()
{
```

```
      if(pos_revendedor == cant)// Si el es el ultimo
            pos_revendedor = -1;
      cant--;
      espera[2].signal(); // Despierta siguiente de la fila
}

void reventa(int id)
{
      pos_revendedor := id;
      espera[id].signal();
      revendedor.wait();
}

end Monitor;

MonitorCola Cola[1..3];

COBEGIN
      Revendedor(1);
      Revendedor(2);
      Revendedor(3);
      Cliente();
      Cliente();
      .........
      Cliente();
COEND.
```