Examen Marzo de 2013

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.**

Problema 1 (36 puntos)

Justifique de forma breve cada respuesta.

1. Qué es un sistema operativo?
2. Cómo se implementa y para que sirve la protección de CPU?
3. Qué entiende por planificador expropiativo (*preemptive scheduling*)?
4. Un sistema cuenta con tres tipos de procesos: de tiempo real, procesamiento por lotes (*batch*) y de usuario. Desarrolle un algoritmo de planificación adecuado para este sistema, mencionando las características más relevantes.
5.
    i. Describa la etapa de ensamblaje (*link-edición*) en el proceso de generar el archivo ejecutable (binario).
    ii. Compare ventajas/desventajas de usar librerías dinámicas o estáticas.
6. Qué entiende por paginación bajo demanda (d*emand paging*)?
7.
    i. Cuándo son ejecutados los algoritmos de reemplazo de marcos (*frames*)?
    ii. Describa el algoritmo de reemplazo LRU (*Least Recently Used*).
8.
    i. Para qué sirven los servicios del sub-sistema de Entrada/Salida que brinda el sistema operativo.
    ii. Describa uno de los principales servicios del sub-sistema de Entrada/Salida.
9. Qué ventajas brinda los sistema RAID (*Redundant Array of Inexpensive Disk*)?

Problema 2 (32 puntos)

Un sistema operativo administra sus archivos en disco utilizando el método de asignación indexada directa con 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 y, que la tabla de inodos, el bitmap, y el disco son globales. Conjuntamente, cada bloque de datos de los directorios tiene 256 entradas de tipo `dir_entry`.

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 con índice block_num en el parámetro `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 con índice `block_num` 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 nombre del archivo referenciado en el parámetro `cam`. Ejemplo, `basename('/home/sistoper/arch.txt') = 'arch.txt'`.

Se pide:

1. Implementar una función que retorna el i-nodo correspondiente al archivo (inodo con de tipo FILE) más grande del sistema:

   ```
   Procedure maxBigFile(Var inode: [0..MAX_INODE_ON_DISK-1]; Var ok : Boolean);
   ```

   En el parámetro `inode` se retorna el índice en la tabla de inodos del inodo de tamaño más grande. En caso que existan mas de un inodo con el máximo tamaño se debe retornar cualquiera de ellos. En el parámetro `ok` se retorna el éxito en la ejecución de la operación.
2. Implementar un función que determine si un directorio esta lleno o no.

   ```
   Procedure IsDirFull(dir: array of char; Var full : Boolean, Var ok : Boolean);
   ```

   El parámetro `dir` representa el camino absoluto del directorio a verificar (Ej. '`/home/sistoper`') y el parámetro `full` retorna verdadero en caso que el directorio tenga todas sus entradas ocupadas. En el parámetro `ok` se retorna el éxito en la ejecución de la operación.

3. Implementar una función que renombra un archivo o directorio dentro de un directorio:

   ```
   Procedure rename(origen: array of char; nuevo : array of char;; Var ok : Boolean);
   ```

   El parámetro `origen` representa el camino absoluto del archivo original (Ej. '`/home/sistoper/origen.txt`') y el parámetro `nuevo` representa el nuevo nombre del archivo o directorio (Ej. '`destino.txt`'). En el parámetro `ok` se retorna el éxito en la ejecución de la operación.

Solución:

```
Procedure maxBigFile(Var inode: [0..MAX_INODE_ON_DISK-1]; Var ok : Boolean)
var i, maxTam: Integer;
begin
      maxTam := 0; ok := false;
      //Si no hay ningún archivo retorna false
      for i = 0 to MAX_INODE_ON_DISK do
            if ((TI[i].used) && (TI[i].type == FILE) && (TI[i].size >= maxTam))
                  // Considero el caso en que todos los archivos estén vacíos.
                  // Devuelvo el último.
                   inode := i;
                   maxTam := TI[i].size;
                   ok := true;
            end
      end
end

Procedure IsDirFull(dir: array of char; Var full : Boolean, Var ok : Boolean)
var nro_origen: [0..MAX_INODE_ON_DISK];
      entryIdx, blockIdx, size: Integer;
      buff: array [0..255] of dir_entry;
bgin
      getInode(dir, nro_inodo, ok);
      if(not ok)
            return;
      end
      if(TI[nro_inodo].type == FILE)
            ok := false;
            return;
      end
      if( TI[inode].size < 5*4096 )
            ok := true;
            full := false;
            return;
      end
      full := true;
      entryIdx := 0;
      blockIdx := 0;
      size := 0;
```

```
      while ( full AND size < TI[nro_inodo].size )
            if(entryIdx == 0)
                  readBlock(D, TI[nro_origen].data[blockIdx], buff,ok);
                  if(not ok)
                        return;
                  end
            end
            full := full AND buff[entryIdx].used;
            entryIdx = entryIdx + 1;
            if(entryIdx == 256)
                  entryIdx := 0;
                  blockIdx := blockIdx + 1;
            end
            size := size + sizeof(dir_entry);
      end
      ok = true;
end

Procedure rename(origen: array of char; nuevo : array of char;; Var ok : Boolean)
var dirPath: array of char;
      nombre_viejo: array of char;
      padre_inodo: [0..MAX_INODE_ON_DISK-1];
      entryIdx, blockIdx, size: Integer;
      encontreBlock, encontreEntry, encontreInodo: Integer;
      buff: Array [0..255] of dir_entry;
begin
      dirname(origen, dirPath);
      basename(origen; nombre_actual);

      getInode(dirPath, padre_inodo, ok);
      if(not ok)
            return;
      end

      if(nuevo == nombre_actual)
            ok := true;
            return;
      end
```

```
      entryIdx := 0; blockIdx := 0; size := 0;
      encontreEntry := -1; encontreBlock := -1; encontreInodo := false;
      while( size < TI[padre_inodo].size  )
            if(entryIdx == 0)
                  readBlock(D, TI[padre_inodo].data[blockIdx], buff,ok);
                  if(not ok)
                        return;
                  end
            end
            if(buff[entryIdx].used)
               //verifico que no exista ya un archivo o directorio con ese nombre
                  if( buff[entryIdx].name == nuevo )
                        ok := false;
                        return;
                  else
                       // verifico que exista el archivo/directorio original
                        if(buff[entryIdx].name == nombre_actual)
                              encontreEntry := entryIdx;
                              encontreBlock := blockIdx;
                              encontreInodo := true;
                        end
                  end
            end
            entryIdx := entryIdx + 1;
            if( entryIdx == 256 )
                  entryIdx := 0;
                  blockIdx := blockIdx + 1;
            end
            size := size + sizeof(dir_entry);
      end
      if(not encontreInodo)
            ok := false;
      end
      readBlock(D, TI[padre_inodo].data[encontreBlock], buff,ok);
      if (not ok)
            return;
      end
      buff[encontreEntry].name := nuevo;
      writeBlock(D, TI[padre_inodo].data[encontreBlock], buff, ok);
end
```

Problema 3 (32 puntos)

Se desea modelar un club de vinos usando mailboxes. El club dispone de 5 Sommeliers que ayudan a los Miembros del club a seleccionar un vino apropiado, un Secretario y un Cajero.

La sede del club tiene capacidad para 15 miembros. Luego de llegar al club los miembros deberán indicar su número de socio al secretario para confirmar si tienen disponible algún vino encargado previamente. En caso afirmativo deberán pasar primero a retirarlo.

Cada miembro deberá ser atendido por el primer sommelier libre. En el caso de que todos los sommeliers estén ocupados, al liberarse uno de ellos deberá atender al siguiente miembro en estricto orden de llegada al club. Una vez seleccionado el vino por el sommelier le entrega la botella al miembro y este se dirige al cajero para pagarla (el cajero solamente atiende a un miembro a la vez). Luego de recibir el dinero el cajero le retorna un ticket al cliente.

Se dispone de las siguientes funciones:

• numero_socio(): NumSocio	Ejecutado por el miembro para saber su número de socio.
• retirar_reserva()	Ejecutado por el miembro para retirar el vino encargado previamente.
• pagar_vino(Botella)	Ejecutado por el miembro para pagar el vino.
• consultar_reserva(NumSocio): boolean	Ejecutado por el secretario para saber si hay alguna reserva para el miembro.
• seleccionar_vino(NumSocio): Botella	Ejecutado por el sommelier para seleccionar un vino para el miembro.
• retornar_ticket(Botella): Ticket	Ejecutado por el cajero para emitir el ticket.

Se pide:

Implementar los procesos Miembro, Secretario, Cajero y Sommelier sin tareas auxiliares.

Es necesario indicar la semántica de los mailboxes utilizados.

Solución: Se utilizarán mailboxes FIFO infinitos, con *send* no bloqueante y *receive* bloqueante.

```
entrar : Mailbox of Integer;
siguiente_cliente : Mailbox of Integer;
mutex_salir : Mailbox of NIL;
secretario_consultar : Mailbox of {NumSocio, Integer};
secretario_respuesta : Array [1..15] of Mailbox of Boolean;

sommelier_atender : Array [1..15] Mailbox of NumSocio;
sommelier_recomendacion : Array [1..15] of Mailbox of Botella;

cajero_atender : Mailbox of NIL;
cajero_cobrar : Mailbox of Botella;
cajero_ticket : Mailbox of Ticket;

procedure miembro()
var posCliente : Integer;
    botella: Botella;
    ticket: Ticket;
    hayReserva: Boolean;
    numSocio: NumSocio;
begin
    numSocio := numero_socio();
    entrar.receive(posCliente);

    secretario_consultar.send(numSocio, posCliente);
    secretario_respuesta[posCliente].receive(hayReserva);
    if(hayReserva)
        retirar_reserva();
    end

    sommelier_atender[posCliente].send(numSocio);
    sommelier_recomendacion[posCliente].receive(botella);

    cajero_atender.receive();
     pagar_vino(botella);
     cajero_cobrar.send(botella);
     cajero_ticket.receive(ticket);
    cajero_atender.send(NIL);

    mutex_salir.receive();
     entrar.send(posCliente);
     siguiente_cliente.send(posCliente);
    mutex_salir.send(NIL);
end

procedure secretario()
var numCliente: NumSocio;
    hayReserva: Boolean;
    posCliente: Integer;
begin
    loop
        secretario_consultar.receive(numcliente, posCliente);
        hayReserva := consultar_reserva(numSocio);
        secretario_respuesta[posCliente].send(hayReserva);
    end
end
```

```
procedure cajero()
var botella: Botella;
    ticket: Ticket;
begin
    cajero_atender.send(NIL);
    loop
        cajero_cobrar.receive(botella);
        ticket := retornar_ticket(botella);
        cajero_ticket.send(ticket);
    end
end

procedure sommelier()
var: posCliente: Integer;
     numSocio: NumSocio;
     botella: Botella;
begin
    loop
        siguiente_cliente.receive(posCliente);
        sommelier_atender[poscliente].receive(numSocio);
        botella := seleccionar_vino(numSocio);
        sommelier_recomendacion[posCliente].send(botella);
    end
end

mein()
var: i: Integer;
begin
    for i = 1 to 15 do
        entrar.send(i);
        siguiente_cliente.send(i);
    end
    mutex_salir.send(NIL);
   cobegin
        cajero;
        miembro;
        sommelier;
        sommelier;
        sommelier;
        sommelier;
        sommelier;
        miembro;
        ....
        miembro;
    end;
end
```