Examen Marzo 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 un ejercicio entero bien hecho y medio más.**

Finalización

- **El examen dura 4 horas.**

Problema 1

Justifique brevemente cada respuesta

1. ¿ Qué entiende por sistema multiprogramado ?.
2. Describa 4 componentes de los componentes más importantes de un sistema operativo.
3. ¿ Qué permiten los llamados al sistema (*system calls*) ?.
4. Sea un sistema que cuenta con los siguientes cuatro procesos con sus respectivos tiempos de ejecución (*burst time*):

Proceso	Tiempo de ejecución
P1	5
P2	4
P3	1
P4	6

   a. Realice un diagrama en el tiempo del uso del procesador para los siguientes planificadores: *FCFS*, *SJF* y *RR* con tiempo de quantum 2.
   b. Calcule el tiempo promedio de espera para los 3 planificadores.
   c. Realice el diagrama para el planificador RR con tiempo de quantum 5 y haga un análisis de cómo se comporta.
5. ¿ Qué entiende por carga dinámica ?.
6. ¿ Qué son las tablas de páginas de un proceso ?.
7. Sea un sistema operativo que implementa paginación bajo demanda con una estrategia de asignación local de marcos (*frames*). Sea un proceso al cual se le asignaron cuatro (4) marcos en memoria principal y que realiza la siguiente secuencia de accesos a sus páginas:

   7 1 2 7 4 5 6 7 3 3 2 7 1 2 6 4 7

   a. Realice un diagrama que muestre en el tiempo el uso de los cuatro marcos si se utiliza un algoritmo de remplazo LRU (*Least Recently Used*) y mencione cuantos fallos de página hubieron.
   b. Realice un diagrama que muestre en el tiempo el uso de los cuatro marcos si se utiliza un algoritmo de remplazo Óptimo (*Optimal*) y mencione cuantos fallos de página hubieron.
8. ¿ Qué ventajas brindan los sistemas RAID ?.

Problema 2

Un sistema operativo administra sus archivos en disco utilizando el método de asignación indexada.

Para esto, se dispone de las siguientes estructuras:

```
type block = array [0..511] of byte;             // 512 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;
  D    : disk;
```

A su vez, se sabe que el directorio raíz es el inodo número 0, que la tabla de inodos y el disco son globales, y que cada bloque de datos de los directorios tiene 32 entradas de tipo `dir_entry`.

a. Implementar una función que retorne la cantidad de bytes utilizados por los archivos (`type == file`) del sistema de archivos.

b. Implementar una función que busque un archivo (`file` o `dir`) dentro de un directorio (no búsqueda recursiva).

```
Procedure searchFile( archivo : Array [1..12] of char; inodo : Integer; Var nro_inodo; Var ok : Boolean);
```

Donde `archivo` es el nombre del archivo a buscar, `inodo` es el número de inodo del directorio donde buscar el archivo, `nro_inodo` es para retornar el número de inodo buscado y `ok` es para retornar si la operación se concretó con éxito o no.

Asumir que se dispone de una función (readBlock) que lee del disco el bloque pasado como parámetro: `readBlock(d : disk; block_num : 0..max_blocks_on_disk; Var buff : block);`

c) Implementar una función que dado un camino absoluto (Ej.: `/home/sistoper/archivo.txt`) retorne el número de inodo correspondiente.

```
Procedure getInode(cam : array of char; Var nro_inodo : Integer; Var ok : Boolean);
```

Donde `cam` es el camino absoluto, `nro_inodo` es el número de inodo del archivo referenciado y `ok` es para devolver si la operación se concretó con éxito o no.

Solución:

a.

```
Procedure filesSize(Var size : Integer);
Var i : Integer;
Begin
  size = 0;
  For i = 1 to max_inode_on_disk do begin
    If ((TI[i].used) and (TI[i].type == file)) then
      size += T[i].size;
    End if;
  End for;
End;
```

b.

```
Procedure searchFile( archivo : Array [1..12] of char; inodo : Integer;
Var nro_inodo: Integer; Var ok : Boolean);
Var
  size : Integer;
  block : Integer;
  entry : Integer;
  buff : Array [1..32] of dir_entry;
Begin
  If (ok = (TI[inodo].type == dir)) then
    size = 0; block = 0; ok = false;
    While (not ok) and (size <= TI[inodo].size) do begin
      If ((size mod 512) == 0) then
        readBlock(d, TI[i].data[block], buff);
        entry = 1;
        block++;
      End if;
      If (ok = (buff[entry].used and (buff[entry].name == archivo)))then
        nro_inodo = buff[entry].inode_num;
      End if;
      size += sizeof(dir_entry);  // 16 bytes
      entry = (entry & 31) + 1;
    End while;
  End if;
End;
```

c.

Se asume las funciones basename y dirname que retornan el nombre del archivo y el directorio respectivamente.

Ej.1 basename('/home/sistoper/archivo.txt') retorna 'archivo.txt' y dirname('/home/sistoper/archivo.txt') retorna '/home/sistoper'.

Ej.2 basename('/archive.txt') retorna 'archivo.txt' y dirname('/archive.txt') retorna '/'.

Solución recursiva

```
Procedure getInode(cam : array of char; Var nro_inodo : Integer; Var ok
: Boolean);
Var
  name : array of char;
  dir  : array of char;
  inodo: Integer;
Begin
  If (cam == '/') then
    ok = true;
    nro_inodo = 0;
  Else
    name = basename(cam);
    dir = dirname(cam);
    getInode(dir,inodo,ok);
    If (ok) then
      searchFile(name,inodo,nro_inodo,ok);
    End if;
  End if;
End;
```

Solución iterativa

Se asume implementado el TAD Pila con sus operaciones usuales.

```
Procedure parser(camino: array of char; Var pila : Pila)
Var
  name : array of char;
Begin
  empty(pila);
  While (camino <> "") do
    name := basename(camino);
    push(name, pila);
    dir  := dirname(camino);
  End while;
End;

Procedure getInode(cam : array of char; Var nro_inodo : Integer; Var ok
: Boolean);
Var
  name : array of char;
  pila : Pila;
Begin
  nro_inodo = 0;
  parser(cam,pila);
  If (ok = (not isEmpty(pila))) then
    name = pop(pila);
    If (name<>"/") then
      While ((not isEmpty(pila)) and ok) do
        name := pop(pila);
        searchFile(name,nro_inodo,nro_inodo,ok);
      End while;
    End If;
  End If;
End;
```

Problema 3

La estación de servicio “La Única” expende nueve tipos de combustible (común, súper, gasoil, premium y gasoil especial, alcohol, kerosene, gas, biodisel). Para ello dispone de un único surtidor por tipo de combustible y 4 expendedores.

La estación consta de un único camino de entrada, por lo que los autos que ingresan se forman en una única fila por orden de llegada, sin tener límite la cantidad de autos posibles en la cola. El primer expendedor libre atenderá al primero de la fila de autos.

Se pide:

Modelar en Ada las tareas Auto y Expendedor.

Se dispone de los siguientes procedimientos:

- **nafta_requerida(out: Integer(1..9) tipo_nafta)** Llamada por los autos y que devuelve el tipo de nafta requerida.
- **expender_nafta(in: Integer(1..9) tipo_nafta)** Llamada por los expendedores para poner la nafta en el auto que se está atendiendo.
- **pagar_cobrar()** Deberá ser ejecutado por el auto o el expendedor para que se realice el pago por el combustible comprado (solamente debe ser llamada por una de las tareas).
- **arrancar()** Ejecutada por el auto una vez le cargaron nafta y se realizó el pago.

Nota: Se pueden utilizar tareas auxiliares.

Solución:

```
Task Type Expendedor is
  Entry dar_id(i: in integer(1..4));
  Entry atencion(i: in integer(1..9));
End Expendedor;

Task Body Expendedor is
Var
     mi_id: integer;
     tipo_nafta: integer;
Begin
   accept dar_id(i: in integer(1..4))
      mi_id = i;
   end;
   loop
     accept atencion(tipo_nafta : in integer(1..9))
        admin_surtidor(tipo_nafta).obtener();
        expender_nafta(tipo_nafta);
        admin_surtidor(tipo_nafta).liberar();
        pagar_cobrar();
     end accept;
     admin_expendedores.fin_expendedor(my_id);
   End loop;
End Expendedor;
```

```
Task Auto is
End Auto;

Task Body Auto is
Var
  expendedor: integer(1..4);
  nafta: integer(1..9);
Begin
  nafta_requerida(nafta);
  admin_expendedores.entrar(expendedor);
  expendedores(expendedor).atencion(nafta);
  arrancar();
End;

Task admin_expendedores is
  Entry entrar(out: expendedor integer(1..4));
  Entry fin_expendedor(in: expendedor integer(1..4));
End admin_expendedores;

Task Body admin_expendedores is
Var
  l_expendedores: array[1..4] of boolean;
  u_expendedor : integer;
  libres  : integer;
Begin
     u_expendedor = 1;
     libres = 4;
     for i = 1 to 4
          l_expendedores(i) = true;
          expendedores(i).dar_id(i);
     endfor
  loop
    select
      when libres > 0 =>
          accept entrar(out: expendedor integer(1..4))
               expendedor = u_expendedor;
          end;
          l_expendedores(u_expendedor) = false;
          u_expendedor = (u_expendedor % 4) + 1;
          libres = libres - 1;
      or accept fin_expendedor(in: expendedor integer(1..4))
                    l_expendedores(expendedor) = true;
          end;
          libres = libres + 1;
    end select;
    if(libres > 0)
       while not(l_expendedores(u_expendedor)) do  -- busca exp libre
          u_expendedor = (u_expendedor % 4) + 1;
       endwhile;
    endif
  end loop;
End admin_expendedores;

Task type admin_surtidor is
     Entry Obtener();
     Entry Liberar();
End admin_surtidor;
```

```
Task body surtidor is
Begin
     Loop
          Accept Obtener();
          Accept Liberar();
     End Loop;
End surtidor

Var expendedores: Array(1..4) of Expendedor;
Var admin_surtidor: Array(1..9) of Surtidor;
```