Examen Agosto de 2003

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

Para las siguientes partes, conteste justificando adecuadamente, cada una de las preguntas.

- **Parte 1**

a. Describa los principios y beneficios en que se sustenta la multiprogramación, mencionando que soporte de hardware requiere.

b. Describa un sistema de planificación de procesos basado en colas multinivel. Indique que ventajas y desventajas presenta.

- **Parte 2**

a. Describa las diferentes modalidades de implementación hilos en un sistema operativo indicando que ventajas y desventajas presenta cada uno.

b. En sistemas con hilos implementadas en el núcleo, un proceso multihilado, ¿cuántas pilas debe tener?

- **Parte 3**

a. Describa los diferentes estados de un proceso, así como las principales colas asociadas al planificador de los mismos.

b. Detalle y explique el cometido de los principales componentes del bloque descriptor de proceso (PCB).

==========

Solución:

Ver libro

==========

Problema 2

Se dispone de un sistema operativo que almacena sus datos en un sistema de archivos tipo XSFAT (eXtended Secure FAT). Este sistema tiene algunas modificaciones con respecto al clásico FAT, ya que:

- Soporta archivos y directorios jerarquizados
- Permite almacenar los datos en múltiples discos duros. Esto es, un archivo puede estar disperso a lo largo de múltiples dispositivos
- Los archivos y directorios del mismo tienen asociados un conjunto de usuarios los cuales podrán operar con los mismos (altas, bajas, lecturas y escrituras)
- Los discos poseen sectores de 256 bytes. Si un archivo utiliza parte de un sector, entonces ese sector es inaccesible para otro archivo.

Se dispone de las siguientes funciones y procedimiento auxiliares:

- **busca_libre(in tabla): integer**
  Dada una tabla (según su definición) retorna un índice en la misma donde exista una entrada libre

- **parse(in string, out string[ ], out largo)**
  Dado un string que representa una ruta, devuelve un arreglo de strings conteniendo los componentes de la ruta, asumiendo que estos componentes están separado por "/".

  Por ejemplo, al invocar **parse("/dir1/dir2/arch1", salida, largo)**, obtendremos
  salida = { "dir1", "dir2", "arch1" } y largo = 3

- **read y write**
  Puede asumir que existe una función "read" para lectura de datos del disco, que recibe la dirección física donde estos se encuentran y retorna un arreglo de bytes con los datos allí presentes. Análogamente se puede suponer lo mismo para la función "write"

Nota: Puede asumir también que el directorio raíz se encuentra en una posición fija definida por usted.

Se pide:

a) Definir las estructuras de datos de la XSFAT, de manera tal que los requerimientos anteriores puedan ser cumplidos

b) Implementar el procedimiento **xs_create** que crea un archivo en disco. Este tiene el siguiente prototipo:

   xs_create(in name: string, in data: byte[], in users: string[100])

   - name: Nombre completo del archivo, incluyendo la ruta completa
   - data: Es un array de bytes con los datos del archivo
   - users: Es el array de usuarios que tienen permiso para trabajar con el archivo

==========

Solución:

a)

```
type address = record
                       sector: integer;
                       volumen: integer;
                  end;

type files = record
                    libre: boolean;
                    nombre: string;
                    usuarios: string[100];
                    tipo: (archivo, directorio);
                    comienzo: address;
               end;
```

- Hay un registro de este tipo en el disco para cada archivo o directorio

```
type xsfat = array [1..MAX_NUM_DISP][1..MAX_NUM_SECTORES] of address;
```

- El primer índice es el número de dispositivo y el segundo es el número de sector dentro del dispositivo.
- Cuando el sector está vacío ambos campos tienen -1 y cuando es el último ambos campos del address tienen -2

```
type hijos = record
                  libre: boolean;
                  nodo: integer;
             end

type directorio = record
                        padre: integer;
                        contenido: hijos[MAX_NUM_HIJOS]
                    end
```

- Cuando el archivo es un directorio el contenido del mismo es un dato de tipo directorio
- Tiene el índice en la tabla de archivos del directorio padre y en la tabla *contenido* están los índices de los directorios y archivos hijos

b)

```
var fat: xsfat;
var nodos: files[MAX_NUM_FILES]

procedure xs_create(in name: string, in data: byte[],
                    in users: string[100])
var    i, pos, nodo: integer;
       dir: directorio;
       sector: address;
       nomArch: string;

begin
       i := translate(name, nomArch);   // da el nodo del directorio padre donde se va a
                                        //crear el archivo y el nombre del archivo
       dir := file_read(nodos[i].comienzo);  // cast implícito
       pos := busca_libre(dir.contenido);
       nodo := busca_libre(nodos);
       dir.contenido[pos].libre := false;
       dir.contenido[pos].nodo := nodo;
       file_write (nodos[i].comienzo, dir);
```

```
	nodos[nodo].libre := false;
	nodos[nodo].nombre := nomArch;
	nodos[nodo].usuarios := users;
	nodos[nodo].tipo := archivo;
	sector := busca_libre(fat);
	nodos[nodo].comienzo := sector;

	file_write (sector, data);
end;


procedure translate (s: string, var nom: string): integer;
var	nodo, largo, i: integer;
	tokens: string[];
	dir: directorio;

begin
	nodo := 0;  // directorio raíz
	parse (s, tokens, largo);
	for i:= 1 to largo-1 do
	begin
		dir := file_read(nodos[nodo].comienzo);
		nodo := lookup (tokens[i], dir);
		if nodos[nodo].tipo <> directorio then
			ERROR;
	end;
	nom := tokens[largo];
	return nodo;
end;


procedure lookup (s: string, dir: directorio): integer;
var	i: integer;

begin
	i :=  1;
	while   nodos[dir.contenido[i].nodo].nombre <> s do
		i := i+1;
	return dir.contenido[i].nodo;
end;


procedure file_read (ad: address): byte[];
var	buffer: byte[];
	offset: integer;

begin
	offset := 0;
	repeat
		read (ad.volumen, ad.sector, buffer, offset, 256);
		// leo siempre sectores enteros porque se que no hay nada más en ellos
		offset := offset + 256;
		ad := fat[ad.volumen][ad.sector];
	until fat[ad.volumen][ad.sector].volumen = -2
	return buffer;
end


procedure file_write (ad: address, data: byte[]);
var	i, cota, offset: integer;
	ad2: address;
begin
	offset := 0;
	if data.largo mod 256 = 0 then
		cota := data div 256;
	else
```

```
            cota := data div 256 + 1;

      for i := 1 to cota do
      begin
            write(ad.volumen, ad.sector, data, offset, 256);
            // escribo de a sectores enteros
            offset := offset + 256;

            if i <> cota then
            begin
                  ad2 := busca_libre(fat);
                  fat[ad.volumen][ad.sector] := ad2;
                  ad := ad2;
            end
            else
            begin
                  fat[ad.volumen][ad.sector].volumen := -2;
                  fat[ad.volumen][ad.sector].sector := -2;
            end
      end;
```

Write recibe la dirección física (volumen y sector dentro del volumen), el arreglo de bytes con los datos a grabar, la posición dentro de este arreglo donde están los datos a guardar y el largo de los datos a grabar.

Read recibe la dirección física (volumen y sector dentro del volumen), un arreglo donde se guardarán los datos leídos, la posición en este arreglo donde se comenzarán a guardar los datos y el largo de los datos a guardar.

==========

Problema 3

Una estación de servicio tiene un gran tanque de combustible con tres surtidores, los cuales podrán ser elegidos por los autos al azar. Por razones de seguridad no deberán existir autos abasteciéndose mientras el gran tanque es recargado. Un camión cisterna pasa cada cierto tiempo aleatorio por la estación a fin de recargar dicho tanque.

Además la estación dispone de dos cajas registradoras en la cuales debe hacerse el pago del combustible cargado. Para esto, una vez finalizada la carga por parte de los vehículos, estos deberán dirigirse a la caja con menos vehículos esperando para pagar y allí realizar el pago.

Por otro lado hay un inspector que le pedirá cada cierto tiempo a los surtidores cuantos litros han expedido y a las cajas cuanto han cobrado. Por razones de seguridad los datos de cuanto se cobro o cuanto se expidió de combustible no podrán pasar por ninguna tarea auxiliar. Es decir que debe ser devueltos por los mismos monitores que ejecutan surtir_combustible y cobrar_combustible respectivamente.

Se desea modelar los autos, el camión cisterna, los surtidores y las cajas.

Notas:

- **Los surtidores deberán cargar simultáneamente tanto entre ellos como con la cobranza. Las cajas también deberán poder cobrar simultáneamente entre ellas.**
- **La cobranza no debe suspenderse por la llegada del camión cisterna**
- **Se ENFATIZA el hecho de que el auto debe ir a la caja con menos autos esperando para pagar.**
- **Se recuerda que solamente los monitores que ejecutan surtir_combustible o cobrar_combustible deben conocer la cantidad total de combustible cargado y la cantidad cobrada respectivamente.**
- **La cantidad de autos es infinita y ejecutaran el siguiente código al llegar a la estación:**

```
procedure AUTO is
begin
     litros = random(40);
     precio = cargar_combustible(litros);
     pagar_combustible(precio);
end;
```

- **Existe un unico camion cisterna, que ejecuta el siguiente codigo**

```
procedure CISTERNA is
begin
     loop
          esperar(random(100));
          recargar_el_gran_tanque();
     endloop
end;
```

- **Se dispone de las siguientes funciones y procedimiento auxiliares:**
  - **surtir_combustible(in: litros, out: precio), que deberá ser ejecutado por el surtidor.**
  - **cobrar_combustible(in: precio), que deberá ser ejecutado por la caja.**
  - **llenar_gran_tanque(), que recarga a su máximo el contenido del gran tanque y puede ser invocado por recargar_el_gran_tanque o cualquier procedimiento que crea conveniente.**
- **Para el resto pueden utilizarse los monitores, procedimientos y estructuras auxiliares que sean necesarias**

Se pide implementar usando monitores a las cajas y surtidores así como también los siguientes procedimientos:

- `cargar_combustible`
- `pagar_combustible`
- `recargar_el_gran_tanque`

junto con todas las estructuras, procedimientos y funciones que sean necesarios

No es necesario implementar el inspector sino simplemente explicitar como interactúa con las cajas y los surtidores

==========

Solución:

```
// Modelo cada surtidor con un monitor.
type monitor surtidores;
carga_gt : boolean;
esperar  : condition;
lexp     : integer;

      procedure intencion_carga_gran_tanque();
      begin
            carga_gt := true;

      end intencion_carga_gran_tanque;

      procedure fin_carga_gran_tanque();
      begin
            carga_gt := false;
            esperar.signal;
      end fin_carga_gran_tanque;

      procedure cargar(in litros : integer; out precio : real);
      begin
            if (carga_gt) then
                  esperar.wait;
            surtir_combustible(in litros : integer; out precio : real);
            lexp := lexp + litros;
            esperar.signal;
      end cargar;
```

```
procedure litros_expedidos(out litros : integer);
      begin
            litros := lexp;
      end litros_expedidos;

begin
      carga_gt := false;
      lexp := 0;
end surtidores;


// Modelo cada caja con un monitor.
monitor caja is
   int total_cobrado;

   // cobra el combustible cargado por el auto
   procedure cobrar(in precio: integer) begin
      cobrar_combustible(precio);
      total_cobrado = total_cobrado + precio;
   end

   // devuelve al inspector el total de plata
   procedure dinero_cobrado(out plata: integer) begin
      plata := total_cobrado;
   end

begin
   total_cobrado := 0;
end monitor // caja
```

```
// Para controlar el acceso a las cajas, utilizamos un administrador que controle el
// largo de las colas formadas
monitor admin_cajas is
   integer largo_caja_1, largo_caja_2;
   condition caja_1, caja_2;

   // El auto avisa que empieza a cobrar, el monitor le devuelve la cola a la
   // que va. Si estan ocupadas las cajas, se lo hace esperar
   procedure cobrar_empezar(out nro_caja: integer) begin
      // primero determino el largo de las colas para ver a que caja va.
      if largo_caja_1 > largo_caja_2 then
         nro_caja = 2;
         largo_caja_2++;
      else
         nro_caja = 1;
         largo_caja_1++;
      endif


   end

   // El auto avisa que termino de cobrar en una caja.
   // Se marca la caja como libre y se permite entrar a otro auto
   procedure cobrar_terminar(in nro_caja: integer) begin
      if nro_caja == 1 then
         largo_caja_1--;
      else
         largo_caja_2--;
      endif
   end

begin
   largo_caja_1 = 0;
   largo_caja_2 = 0;
end monitor // admin_cajas

// Programa principal
caja            mon_cajas[1..2]
surtidor        mon_surtidor[1..3]
admin_cajas     mon_admin_cajas;


procedure pagar_combustible(in precio: integer) begin
   integer nro_caja;

   mon_admin_cajas.cobrar_empezar(nro_caja);  // Buscamos una caja
   mon_cajas[nro_caja].pagar(precio);         // Pagamos la nafta

   mon_admin_cajas.cobrar_terminar(nro_caja); // Avisamos que terminamos

end

procedure recargar_gran_tanque();
begin
   for i := 1 to 3 do
         # Si hay algún auto cargando entonces se bloquea debido a que el auto carga
         #dentro de un procedimiento del monitor
        mon_surtidor[i].intencion_carga_gran_tanque;

   llenar_gran_tanque();
   for i := 1 to 3 do
       mon_surtidor[i].fin_carga_gran_tanque;
end recargar_gran_tanque();
```

```
function cargar_combustible(in litros : integer) out precio : real;
integer nro_surtidor, precio;
begin
       mon_surtidor[random(3)].cargar(litros,precio);
       return precio;
end cargar_combustible;
```

Los inspectores tendrán que invocar a los procedimientos del monitor mon_surtidor[Numero_surtidor_deseado].litros_expedidos (litros) que le devolverá en la variable entera litros la cantidad surtida por el surtidor Numero_surtidor_deseado y mon_cajas[nro_caja].dinero_cobrado(dinero) que le devolverá en la variable entera dinero la cantidad cobrada por la caja nro_caja

==========