Examen Febrero 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 es y cuáles ventajas propone un sistema de tiempo compartido?
2. Realice un esquema con los estados por los que puede pasar un proceso y sus transiciones.
3. Sea un sistema con paginación bajo demanda y que soporta hilos a nivel del núcleo y sean los registros Program Counter y Page Table Base Register.
    a. Cuáles de los registros anteriores son modificados al realizar un cambio de contexto entre dos hilos de un mismo proceso.
    b. Cuáles de los registros anteriores son modificados al realizar un cambio de contexto entre dos hilos que pertenecen a procesos distintos.
4. Sea un sistema operativo con un planificador con prioridades estáticas.
    a. ¿Qué problema presenta este tipo de planificador?
    b. Brinde una solución al problema mencionado en la parte anterior.
5. En general los sistemas operativos modernos de propósito general dividen el espacio virtual de los procesos en varios componentes. Mencione estos componentes y su función.
6. Compare un esquema de paginación que utiliza varios niveles de tabla de páginas (digamos 4 niveles) contra un sistema de paginación que utiliza solamente un nivel para la tabla de página.
7. ¿Qué es la TLB, cuál es su función y qué sucede con ella cuando hay un cambio de contexto entre dos procesos distintos?
8. ¿Qué entiende por memoria SWAP?
9. En el subsistema de Entrada/Salida, describa dos métodos que se pueden utilizar para efectuar una E/S.

Problema 2

Un sistema operativo maneja sus archivos en disco utilizando las siguientes estructuras:

```
type  directorio = array [1..max_files_on_disk] of record
                     used : boolean {entrada usada o no}
                     file_name : array [1..8] of char;
                     file_ext : array [1..3] of char;
                     informacion : array [1..100] of char;
                     comienzo : integer; {dirección del primer sector asignado}
                end;
type mapa = array [1..sectors_in_disk] of integer;
type sector = array [1..1024] of bytes;
Var
   disco: array [1..sectors_in_disk] of sector;
```

Cada elemento del mapa corresponde a un sector del disco. Los sectores son de tamaño fijo de 1024 bytes.

Cuando ese sector no está utilizado el elemento correspondiente del mapa contiene el valor cero.

En caso de que el sector este asignado a un archivo pero no sea el último del mismo, contiene la dirección dentro del mapa del siguiente sector asignado

a dicho archivo.

Cuando es el último sector asignado a un archivo, contiene FFFF.

Se pide:

a) Escribir una función que devuelva el tamaño en bytes que ocupa un archivo en disco.

```
Procedure TamañoArchivo(fat : mapa; dir : directorio; nombre : array [1..8] of char; ext
: array [1..3] of char; Var tam : integer; Var ok : boolean);
```

b) Especificar y escribir una función que cree un archivo nuevo en el directorio de un tamaño dado e inicialice los sectores con el valor 0. Utilizar un algoritmo que vaya asignando los bloques mediante una estrategia de *worst fit*. Especifique parámetros de entrada y salida.

Asuma que siempre habrá sectores disponibles para realizar la asignación pedida.

c) Escribir una función que devuelva si el sistema de archivos está en un estado consistente.

```
Procedure ChkDisk(fat : mapa; dir : directorio; var ok : boolean);
```

Nota: Se puede asumir que el disco es accedido globalmente.

Solución:

Solo se controlará que el archivo exista dentro del directorio.

```
Const SECTORSIZE = 1024;
Procedure TamañoArchivo(fat: mapa; dir : directorio; nombre : array
[1..8] of char; ext : array[1..3] of char; Var tam : integer; Var ok :
boolean)
Var
    i        : intenger;
Begin
     ok = false; i = tam =0;
     while ((not ok) and (i < max_files_on_disk)) do begin
          i++;
          ok = (dir[i].used and (nombre == dir[i].nombre) and
                                          (ext == dir[i].ext));
     end;
     if (ok) then
          pos = dir[i].comienzo;
          tam = SECTORSIZE;
          while (fat[pos] != 0xFFFF) do begin
               pos = fat[pos];
               tam += SECTORSIZE;
          end;
     endif
End;
```

Es válido también retornar la cantidad de sectores y no el tamaño.

b.

```
Procedure WF (fat : map; var inicio : integer; var cantidad : integer);
Var
  i         : integer;
  cantactual : integer;
Begin
  inicio = 1; cantidad = cantactual = 0;
  for i = 1 to sectors_in_disk do begin
    if (fat[i] == 0) then  { está libre }
       cantactual++;
    else begin
       if (cantactual > cantidad) then
          cantidad = cantactual;
          inicio = i - cantactual;
       else
          cantactual = 0;
    endif
  end;
  if (cantidad == 0) then cantidad = cantactual; { caso: toda fat libre }
End;
```

Solo se controlará que tengamos lugar en el directorio para un nuevo archivo.

```
Procedure CrearArchivo(var dir : directorio; var fat : mapa; nombre
array[1..8] of char; ext : array[1.3] of char; info : array[1..100] of
char; nrosectores : integer; var ok Boolean);
Var
     i        : integer;
     j        : integer;
     encontre : boolean;
     primero  : boolean;
     ultimo   : integer;
Begin
     ok = false; i = 1;
     while (i <= max_files_on_disk) and dir[i].used do
          i++;
     if (i <= max_files_on_disk) then  { encontre libre }
          dir[i].used = ok = true;
          dir[i].file_name = nombre;
          dir[i].file_ext = ext;
          dir[i].informacion = info;
          primero = true;
          Repeat
               WF(fat, inicio, cantidad);
               if (primero)
                    dir[i].comienzo = inicio;
                    primero = false;
               else
                    fat[ultimo] = inicio;
               endif
               if (cantidad > nrosectores)
                    cantidad = nrosectores;
               endif
               nrosectores -= cantidad;
               ultimo = incio + cantidad;
               fat[ultimo] =  0xFFFF;
               zero(disco[ultimo]);
               for j = (ultimo - 1) downto inicio do begin
                    fat[j] = j + 1;
                    zero(disco[j]);
               end;
          until (nrosectores = 0);
       endif
End;
```

c.

Solo se controlará para los archivos usados que no se tenga un loop y que no esté trunco (la recorrida de la lista termine en un sector libre).

```
Procedure ChkDisk (fat : mapa; dir : directorio; var ok : boolean)
Var
  i          : integer;
Begin
  ok = true;
  while (ok or (i <= max_files_on_disk)) do begin
     if (dir[i].used) then
           ok = checkFileFAT(fat,dir[i].comienzo);
     endif
     i++;
  end;
End;

Function checkFileFAT(fat : mapa; i : integer) : Boolean
Var
     cant : integer;
     ok : Boolean;
     trunc : Boolean;
Begin
     cant = 0;
     ok = trunc = false;
     while ((not ok) and cant<=sectors_in_disk and (not trunc)) do begin
           ok = (fat[i] == 0xFFFF);
           trunc = (fat[i] == 0x0);
           i = fat[i];
           cant++;
     end;
     return ok;
End;
```

Otra opción para verificar loops podría ser llevar una estructura con los sectores ya recorridos y controlar en cada paso si ese sector ya se visitó.

Problema 3

Se desea modelar usando monitores el recuento de votos de una mesa de votación de un balotaje que cuenta con un presidente de mesa, un secretario y cuatro delegados partidarios. El presidente irá abriendo los sobres, poniendo las papeletas sobre la mesa hasta un máximo de 10 y llevando la cuenta de los votos emitidos a cada uno de los dos candidatos presentados. Los delegados partidarios mirarán en orden los votos que están en la mesa e irán llevando su propia cuenta de los votos. Cuando el delegado verifica cada una de las papeletas deja una marca delante de ella. Si es el cuarto delegado en verificar el voto deberá avisarle al presidente para que retire la papeleta verificada de la mesa y seguirá verificando los votos siguientes sin necesidad de esperar a que el presidente de mesa lo retire.

Cada cierto tiempo el secretario les solicitará el conteo actual de votos al presidente de mesa y a cada uno de los delegados. Si los conteos no coinciden el recuento se detiene y se comenzará nuevamente más adelante (no es necesario modelar este nuevo recuento)

Los delegados se fijarán si hay una solicitud del secretario inmediatamente después de verificar cada papeleta, quedándose bloqueados hasta que el secretario verifique los números.

Los votos en blanco o anulados no se contabilizan. El conteo termina cuando no hay más votos pero no es necesario modelarlo.

Se dispone de las siguientes funciones:

- abrirSobre(): papeleta Ejecutado por el presidente. Solo retorna papeletas válidas
- verificarPapeleta(papeleta): {1,2} Ejecutado por los delegados y el presidente
- retirarPapeleta() Ejecutado por el presidente. Retira una papeleta ya verificada.
- cancelarRecuento() Ejecutado por el secretario
- esperaSecretario() Ejecutado por el secretario mientras no esta verificando el conteo
- verificarNumeros(array [1..5][1..3] of integer): boolean

  Ejecutado por el secretario. Recibe los conteos de votos del presidente y los 4 delegados para cada uno de los candidatos junto con la posición en la mesa del último voto examinado. Retorna true si los números coinciden.

Solución:

```
presidente()
{
      int[1..3] cont;

      while(true)
      {
            Papeleta p = abrirSobre();
            int voto = verificarPapeleta(p);
            cont[voto]++;
            while(Mesa.deboRetirar())
            { // Retiro papeletas vistas por todos
                  retirarPapeleta();
                  Mesa.retirePapeleta();
            }
            if(Mesa.entregar(p))
            {
                  Mesa.misDatos(5, cont);
            }
      }
}

delegado(int nro)
{
      int[1..3] cont;
      int pos = 0;

      while(true)
      {
            bool darDatos = Mesa.recibir(pos, p);
            pos++;
            int voto = verificarPapeleta(p);
            cont[voto]++;
            if(darDatos)
            {
                  cont[3] = pos%10;
                  Mesa.misDatos(nro, cont);
            }
      }
}

secretario()
{
      while(true)
      {
            esperaSecretario();
            conts = Mesa.quieroVerificar();
            if(!verificarNumeros(conts))
            {
                  cancelarRecuento();
            }
      }
}
```

```
Monitor Mesa
{
      Papeleta papeletas[10];
      int cant = 0;
      int pos = 0;
      int aLiberar = 0;
      int marcas[10];
      int[1..5][1..3] conts;  // Contiene los conteos de votos que el
                              // secretario luego utilizara para verificar numeros

      boolean entregar(Papeleta p)
      {
            papeletas[pos % 10] = p;
            pos++;
            VotoNuevo.signal();
            cant++;
            return SecretarioQuiereVerificar;
      }

      boolean deboRetirar()
      {
            if(aLiberar == 0 && cant == 10)
            { // Si la mesa esta llena de papeletas
                  LiberoMesa.wait();
            }
            return (aLiberar != 0);
      }

      void retirePapeleta()
      {
            aLiberar--;
            cant--;
      }

      boolean recibir(int rpos, Papeleta p)
      {
            if(rpos == pos)
            { // Si ya mire todas las papeletas de la mesa
                  VotoNuevo.wait();
                  VotoNuevo.signal();
            }
            marcas[rpos % 10]++;
            p = papeletas[rpos % 10];
            if(marcas[rpos % 10] == 4)
            { // Si soy el cuarto delegado en verificar
                  marcas[rpos % 10] = 0;
                  aLiberar++;
                  LiberoMesa.signal();
            }
            return SecretarioQuiereVerificar;
      }
```

```
      int[1..4,1..3] quieroVerificar()
      {
            SecretarioQuiereVerificar = true;
            SecretarioEspera.wait();
            DatosListos.signal();
            return conts;
      }

      void misDatos(int nro, int[] cont)
      {
            if (nro == 5) {
                  cont[3] = pos % 10;
            }
            conts[nro] = cont;
            nroConts++;
            if(nroConts == 5)
            {
                  nroConts = 0;
                  SecretarioQuiereVerificar = false;
                  SecretarioEspera.signal();
            }else
            {
                  DatosListos.wait();
                  DatosListos.signal();
            }
      }
}

cobegin
      presidente();
      delegado(1); delegado(2);
      delegado(3); delegado(4);
      secretario();
coend
```