Examen Febrero 2009

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.**

Justificación

- **Toda solución de un problema deberá estar debidamente justificada**
- **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) Describa los sistemas de multiprocesadores simétricos y asimétricos.

2)
   a) Mencione los servicios básicos que debe brindar un sistema operativo.

   b) ¿ De qué forma se los brinda a los procesos a nivel de usuario ?

3) Describa los estados de los procesos y sus transiciones.

4) ¿ En qué tipos de sistemas se pueden utilizar planificadores expropiativos (*preemptive scheduling*)?

5) ¿ Qué entiende por memoria virtual ?

6) Sea un sistema de 32 bits que maneja memoria virtual con paginación. Las direcciones virtuales se componen de 20 bits para direccionar la página y 12 para el desplazamiento.
   a) ¿ De que tamaño son las páginas ?

   b) ¿ De que tamaño son las frames ?

   c) Si las entradas de la tabla de páginas son de 4 bytes y cada tabla es del tamaño de una página, ¿ Cómo se distribuye la componente de 20 bits para direccionar las páginas ?

7) Describa la utilidad de la TLB (Translation Look-Aside Buffer).

8) ¿ Qué deficiencias sufre la segmentación frente a la paginación ?

9) ¿ Qué ventajas ofrecen los sistemas RAID (*Redundant Array of Inexpensive Disk*) ?

10) Describa dos métodos para efectuar operaciones de E/S y de un ejemplo para que tipo de sistema es útil.

Solución:

Ver teórico y/o notas del curso

Problema 2

Considere un sistema de archivos, donde estos se almacenan en secciones contiguas de segmentos. Para un archivo que se encuentra abierto, se ha creado en memoria la siguientes estructura de datos, apuntada por el puntero **FilePointer^**

```
FilePointer := Record
      Extenciones:  Array [0..MAX_EXTENTS-1] of Extencion;
      PosicionActual: Integer;
      Buffer: Array [0..TAM_BUF-1] of Bloque
End

Extencion := Record
      CantBloques: Integer;
      Comienzo: Integer;
      Fin: Integer;
End

Bloque := Array [0..TAM_BLOQUE-1] of Byte
```

Algunas aclaraciones con respecto a las estructuras de datos:

- Una extensión representa una secuencia de bloques almacenada físicamente contigua en el disco, desde la posición **Comienzo** hasta la posición **Fin**.
- Un bloque representa una secuencia de **TAM_BLOQUE** bytes que se puede leer/escribir del disco.
- En FilePointer, **PosicionActual** representa la posición del último byte leído en una operación de lectura secuencial. Tiene el valor -1 si el archivo nunca fue leído secuencialmente.
- Buffer es una estructura auxiliar utilizada por el sistema operativo. Siempre se deben leer **TAM_BUF** bloques.
- El tamaño total de un archivo, medido en bloques, es siempre un múltiplo de **TAM_BUF**

Se dispone de las siguientes funciones auxiliares:

- **LeerBloque (NumBloque Integer, CantBloques: Integer) : Array of Bloques**
  Lee la cantidad de bloques indicada por la variable CantBloques a partir del número de bloque NumBloque.

- **LargoArray (Array of T) : Integer**

Devuelve el largo del array indicado como parámetro (no importa el tipo de elemento (T) almacenado por el array)

Se pide:

a) Implementar las siguientes funciones:

- **LeerArchivoSecuencial (Archivo: FilePointer^, CantBytes: Integer) : Array of Character**
- **LeerArchivoDirecto (Archivo: FilePointer^, Posición: Integer, CantBytes : Integer) : Array of Character**

Ambas funciones devuelven una cadena de caracteres con el largo indicado en el parámetro **CantBytes**. La cadena de caracteres esta almacenada en el archivo indicado con el parámetro **Archivo**.

LeerArchivoSecuencial lo hace a partir de la posición **PosicionActual** del archivo.

LeerArchivoDirecto lo hace a partir de la posición indicada con el parámetro **Posición**.

b) Se entiende por problemas de eficiencia, el hecho de realizar recorridas innecesarias de las estructuras de datos o lecturas excesivas de disco.

La solución propuesta para la parte a) presenta problemas de eficiencia?

En caso afirmativo, indique como pueden mejorarse las estructuras de datos para evitar este problema. En caso negativo, indique porque no los presenta.

Notas y aclaraciones:

- Un carácter ocupa un byte.
- LeerBloque siempre lee la cantidad de bloques indicada como parámetro. Nunca lee pedazos de bloque.
- En caso de ser necesario, pueden agregarse funciones auxiliares. No es necesario implementar las funciones auxiliares agregadas, pero si aclarar su funcionamiento.
- Debe realizar el manejo de errores en las funciones pedidas. Seleccione el mecanismo que considere apropiado.
- Puede implementar las funciones solicitadas utilizando pseudocódigo o algún lenguaje que conozca (C, modula, etc.)

```
-----------------------------------------------------------
Solucion:

Ya implementamos la solución con las mejoras de eficiencia para la parte b)

FilePointer := Record
      Extenciones:  Array [0..MAX_EXTENTS-1] of Extencion;
      PosicionActual: Integer;
      BufLeido: Integer;
      Buffer: Array [0..TAM_BUF-1] of Bloque
End

Extencion := Record
      CantBloques: Integer;
      Comienzo: Integer;
      Fin: Integer;
      Desde: Integer ;
      Hasta: Integer ;
End


LeerArchivoDirecto (Archivo: FilePointer^,
                    Posición: Integer,
                     CantBytes : Integer
                     // no considero cruce frontera de buff
) : Array of Character

   LBloque, LBloqOffset : Integer;
   IExtent, IBuffer, IBufOffset : Integer : = 0; // Tmp
   Rslt : Array of Char;

   Begin
   While CantBytes > 0 do // queda por leer
      Begin
      LBloque := Posicion Div TamBloque;
      LBloqOffset := Posicion – LBloque * TamBloque;
      If LBloque >  Extencion[MaxExtents].Hasta then
            Raise ('Lectura posterior a Fin de Archivo');
      While LBloque > Extencion[].Hasta Do IExtent := IExtent + 1;
      // IExtent indica donde leer
```

```
      // Resta cargar el buffer si no tiene ese contenido
      IBuffer :=  LBloque Div TamBuf;
      IBuffOffset := LBloque - IBuffer * TamBuf; // Bloque del Buff
      // asumo CantBloques de cada extención es múltiplo de TamBuf
      If IBuffer <>  BufLeido then // debo leer un Buffer diferente
            Buffer := LeerBloque (IBuffer, TamBuf) // lleno el Buffer
            BufLeido := IBuffer;
      End;
      // si CantBytes pasa del Buffer, itera otra vez
      Rslt := CopioStr(Buffer[IBuffOffset],
            CantBytes); // actualizo CantBytes con saldo no leido
      Posicion := Posicion + Rslt.Size; // NuevaPosicion
      LeerArchivoDirecto := LeerArchivoDirecto.Value + Rslt
      End;
   // Fin
End;

------------------------------------------------------------
```

Problema 3

Se desea modelar un servicio de venta de celulares en el cual los clientes (ilimitados) piden un modelo de celular para ver al vendedor (hay 10 vendedores en el local) el cual les mostrará ese modelo y dos más de mayor costo. Para obtener los celulares los vendedores deberán pedírselos al encargado el cual dispone de un solo celular de cada modelo (la cantidad de modelos de celular no está acotada).

Mientras espera que le muestren los celulares el cliente deberá tomar un café. Una vez que el vendedor le muestra los tres celulares el cliente decidirá cual le gusta. Si el cliente demorara más de cinco minutos en tomar la decisión, el vendedor se retirará del escritorio a cumplir otras tareas cortas. Luego de terminar estas tareas, si el cliente aún no se decidió, esperará hasta que éste termine de decidirse. Los tres celulares de muestra serán devueltos al encargado una vez que se terminó de atender al cliente (el que se vende no es el de muestra).

Notas:

- No se podrán implementar tareas auxiliares
- Las tareas clientes se crean dinámicamente sin poderse identificar dentro de un arreglo de tareas.
- Se deberá prestar especial atención a no "despertar/desbloquear" tareas innecesariamente, ni generar deadlock.
- No es necesario implementar funciones de búsqueda, inserción o borrado en arreglos o listas.
- Se dispone de las siguientes funciones y procedimientos:
    - modeloInicial(): Modelo ejecutada por el cliente. Devuelve el modelo de celular pretendido por el cliente.
    - elegirVendedor(): Integer ejecutada por el cliente para seleccionar que vendedor quiere que lo atienda.
    - tomarCafe() ejecutada por el cliente
    - elegirModelo (m1 in Modelo, m2 in Modelo, m3 in Modelo): Modelo ejecutada por el cliente para elegir el celular.
    - otrosModelos (m1 in Modelo, m2 out Modelo, m3 out Modelo) ejecutada por el vendedor
    - mostrar (m1 in Modelo, m2 in Modelo, m3 in Modelo) ejecutada por el vendedor
    - tareasCortas ejecutada por el vendedor
    - vender (m1 in Modelo) ejecutada por el vendedor

Se pide:

Modelar en ADA las tareas Cliente, Vendedor y Encargado definiendo los arreglos de tareas que correspondan

---

Solucion:

```
Task Type Cliente is
End Task Cliente;

Task Body Cliente is
m1, m2, m3, m: Modelo;
vend: integer;
Begin
  m=modeloInicial();
  vend=elegirVendedor();
  Vendedores[vend].QuieroVer(vend, m);
  tomarCafe();
  Vendedores[vend].EsperoMuestra(m1, m2, m3);
  m=elegirModelo(m1, m2, m3);
  Vendedores[vend].Comprar(m);
End Task;

Task Type Vendedor is
      Entry QuieroVer(v:in Vendedor, m: in Modelo);
      Entry EsperoMuestra(m1: out Modelo, m2: out Modelo, m3: out Modelo);
      Entry Comprar(m: in Modelo);
      Entry EncargadoMuestrasListo();
End Task;

Task Body Vendedor is
mod1, mod2, mod3: Modelo;
id:integer;
Begin
      Loop
            Accept QuieroVer(v, m)
                  id=v;
                  mod1=m;
            End;
            otrosModelos(mod1, mod2, mod3);
            Encargado.DameMuestras(id, mod1, mod2, mod3, listo);
            if not listo
                  Accept EncargadoMuestrasListo;
            endif
            Accept EsperoMuestra(mod1, mod2, mod3)
                  Mostrar(mod1, mod2, mod3);
            End;
            Select
                  Accept Comprar(m)
                        Vender(m);
                  End;
```

```
                OR Delay 5*60
                        tareasCortas;
                        Accept Comprar(m)
                                Vender(m);
                        End;
            EndSelect

            Encargado.DevuelvoMuestras(mod1, mod2, mod3);
      EndLoop
End Task;

Task Encargado is
      Entry DameMuestras(v: in integer, m1: in Modelo, m2: in Modelo, m3: in
Modelo, listo: boolean);
      Entry DevuelvoMuestras(m1: in Modelo, m2: in Modelo, m3: in Modelo);
End Task;

Task Body Encargado is
mod1, mod2, mod3: Modelo;
vend, l: integer;
usados: list of Modelo;
Begin
      Loop
            Select
                        Accept DameMuestras(v, m1, m2, m3, listo)
                              if(usados.contains(m1)  or  usados.contains(m2)  or
                                 usados.contains(m3))
                                    listo = false;
                              else
                                    mod1=m1; mod2=m2;mod3=m3;
                                    l=true; vend = v;
                                    listo=true
                              endif
                        End
                        if l = true
                              usados.add(mod1);
                              usados.add(mod2);
                              usados.add(mod3);
                        else
                              buscados.add(vend, {mod1, mod2, mod3});
                        endif
                OR Accept DevuelvoMuestras(m1, m2, m3)
                              mod1=m1; mod2=m2; mod3=m3;
                        End
                        usados.remove(mod1);
                        usados.remove(mod2);
                        usados.remove(mod3);
                        while(buscados.buscarTripletaLibre(usados, vend, {m1, m2,
                              m3})) // Busca vendedor con tripleta libre
```

```
                              buscados.eliminar(vend, {m1, m2, m3};
                              usados.add(m1);
                              usados.add(m2);
                              usados.add(m3);
                              Vendedores[vend].EncargadoMuestrasListo();
                        endif
            EndSelect
      EndLoop
End Task;

Vendedores: Array (1..10) of Vendedor;
```
