Examen Diciembre de 2011

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 (35 puntos)

Justfique cada una de sus respuestas:

1. Qué es un sistema operativo?
2. Cómo expone un sistema operativo sus servicios a las aplicaciones?
3. Qué entiende que es un programa? un proceso? un thread?
4. Describa el planificador (*scheduler*) de colas multinivel con retroalimentación (*multi-level feed-back queue*).
5. Qué entiende por memoria virtual?
6.
    i. Cuando es ejecutado un algoritmo de remplazo de página?
    ii. Qué dice la anomalía de Belady?
    iii. Describa el algoritmo de segunda chance.
7. Para qué sirve la paginación bajo demanda?
8. Qué entiende por RAID 0?
9. El caching es un servicio que brinda el sistema operativo, en qué subsistema se encuentra? Para qué sirve?

Problema 2 (30 puntos)

Sea un sistema de archivos simplificado que cuenta con las siguientes estructura de datos:

```
const MAX_DIRS = 4096; const MAX_FILES_ON_DIR = 4096; const MAX_SECTORS_ON_DISK = 16384;
type sector = array [0..511] of byte;
type fs = Array [0..(MAX_DIRS-1)] of dir;
type dir = Record
             dir_entry : array [0..(MAX_FILES_ON_DIR-1)] of file;
             used : boolean;
           end;
type file = Record
                    used : boolean;                    // entrada usada o no
                    file_name : array [0..7] of char; // nombre del archivo
                    file_ext : array [0..2] of char;  // extensión del archivo
                    start : integer;                   // direccion de comienzo
                    type : {FILE, DIRECTORY};          // tipo de archivo
                    dir_index : 1..(MAX_DIRS-1);       // indice en fs
                    size : integer;                    // tamaño del archivo en bytes
             end;
type fat = array [0..(MAX_SECTORS_ON_DISK-1)] of -1..(MAX_SECTORS_ON_DISK-1);
type disk = Array [0..(MAX_SECTORS_ON_DISK-1)] of sector;
Var    SA : fs;  // sistema de archivos
       D : disk; // disco
```

Notas generales:

- Las variables D y SA son globales y la información del directorio raíz está en `SA[0]`.
- En la fat el valor 0 representa "fin de archivo" y el -1 "sector libre".
- El campo `dir_index` en file es solo usado cuando el archivo es un directorio.
- Toda operación debe dejar el sistema de archivos en un estado consistente.

Se pide:

1. ¿Cuál es la cantidad máxima de archivos que pueden existir en este sistema? (5 pts.)
2. Implementar una función que cree un directorio con la siguiente definición:
   `Function createDir(d_padre: 0..(MAX_DIRS-1), name: Array [0..7] of char, ext: Array[0..2] of char): boolean;`
   Donde `d_padre` es el índice del directorio padre donde se quiere crear, `name` es el nombre del directorio a crear, y `ext` es la extensión. La operación debe retornar si finalizó con éxito o no. (10 pts.)
3. Implementar una función que mueva un archivo con la siguiente definición:
   `Function moveFile(d_source: 0..(MAX_DIRS-1); name_s: Array [0..7] of char; ext_s: Array [0..2] of char; d_target: 0..(MAX_DIRS-1); name_t: Array [0..7] of char; ext_t: Array [0..2] of char): boolean;`
   Donde `d_source` es el índice del directorio origen, `name_s` y `ext_s` es el nombre del archivo origen, `d_target` es el índice del directorio destino, y `name_t` y `ext_t` es el nombre del archivo destino. La operación debe retornar si finalizó con éxito o no. Esta operación no es aplicable a directorios. (15 pts.)

Solución Problema 2:

1. **Asumiendo que los archivos vacíos no ocupan un sector: Se pueden tener como máximo 4096 directorios/archivos con 4096 directorios/archivos en cada uno. Por lo que la cantidad máxima de directorios/archivos es de 4096 * 4096 =~ 16M.**

   Asumiendo que los archivos vacíos ocupan al menos un sector: En este caso hay que comparar el número anterior con la cantidad de sectores del disco y tomar el menor de los dos. Como la cantidad de sectores es 16384, entonces la cantidad máxima de archivos/directorios es de 16384.

2. **Function createDir(d_padre: 0..(MAX_FILES_ON_DIR-1), name: Array [0..7] of char, ext: Array[0..2] of char): boolean;**

```
Var ret, existe   : Boolean;
   i, j, k, libre : Integer;


Begin
     ret = existe = false;
     i = 0;
     libre = -1;
     // se busca un lugar libre en el directorio padre y se verifica que
     // ya no existe un archivo (o directorio) con ese nombre
    while (i < MAX_FILES_ON_DIR && !existe) do
        if( SA[d_padre].dir_entry[i].used &&
            SA[d_padre].dir_entry[i].name == name &&
            SA[d_padre].dir_entry[i].ext == ext )
            existe = true;
        else  if(libre == -1 && !SA[d_padre].dir_entry[i].used)
                libre = i;   // obtengo lugar libre
              endif
        end if
        i++;
    end do
   // si encontre lugar libre y no existe archivo con mismo
    // nombre y extensión sigo adelante
    if (libre >= 0 && !existe) then
          j = 0;
          // se busca un lugar libre en la tabla de directorios
          while (SA[j].used) && (j < MAX_DIRS)) do
              j++;
          end do
```

```
                // si existe un lugar libre se crea el directorio
                if (j < MAX_DIRS) then
                        SA[j].used = true;

                                // se marcan como libre todas las entradas del
directorio
                        for (k = 0; k < MAX_FILES_ON_DIR) do
                                SA[j].dir_entry[k].used = false;
                        end for;
                        SA[d_padre].dir_entry[libre].used = true;
                        SA[d_padre].dir_entry[libre].file_name = name;
                        SA[d_padre].dir_entry[libre].file_ext = ext;
                        SA[d_padre].dir_entry[libre].start = -1;
                        SA[d_padre].dir_entry[libre].type = DIRECTORY;
                        SA[d_padre].dir_entry[libre].dir_index = j;
                        SA[d_padre].dir_entry[libre].size = 0;
                        ret = true;
                end if
        end if
        return ret;
    End.
```

3. **Function moveFile(d_source: 0..(MAX_FILES_ON_DIR-1); name_s: Array [0..7] of char; ext_s: Array [0..2] of char; d_target: 0..(MAX_FILES_ON_DIR-1); name_t: Array [0..7] of char; ext_t: Array [0..2] of char): boolean;**

```
Var i, j, libre : Integer;


Begin
  // se busca el archivo a mover
  i = 0;
  while (!(SA[d_source].dir_entry[i].used &&
          (SA[d_source].dir_entry[i].file_name == name_s) &&
          (SA[d_source].dir_entry[i].file_ext == ext_s)) &&
          (i < MAX_FILES_ON_DIR)) do
       i++;
  end do
  // Si no existe el archivo que se desea mover o es un directorio
  // retorno error
  if (i == MAX_FILES_ON_DIR ||
      SA[d_source].dir_entry[i].type == DIRECTORY ) then
          return false;
  end if

  // se busca un lugar libre en el directorio destino y se verifica
  // que ya no existe un archivo (o directorio) con ese nombre
  j = 0;
  libre = -1;
  while (!(SA[d_target].dir_entry[j].used &&
          (SA[d_target].dir_entry[j].file_name == name_t) &&
          (SA[d_target].dir_entry[j].file_ext == ext_t)) &&
          (j < MAX_FILES_ON_DIR)) do
       if( ! SA[d_target].dir_entry[j].used ) then
           libre = j;
       end if
       j++;
  end do
  // si j < MAX_FILES_ON_DIR: ya existe un archivo con ese nombre
  if (j < MAX_FILES_ON_DIR) then
          return false;
  end if
```

```
        // optimización: en caso que el directorio origen y destino
        // sean el mismo en realidad se está renombrando el archivo
        if( d_source == d_target ) then
                SA[d_source].dir_entry[i].file_name = name_t;
                SA[d_source].dir_entry[i].file_ext = ext_t;
                return true;
        end if
        // si libre igual a -1 no: el directorio está lleno
        if (libre == -1) then
                return false;
        end if

        // se copian los datos
        SA[d_target].dir_entry[libre].used = true;
        SA[d_target].dir_entry[libre].file_name = name_t;
        SA[d_target].dir_entry[libre].file_ext = ext_t;
        SA[d_target].dir_entry[libre].start = SA[d_source].dir_entry[i].start;
        SA[d_target].dir_entry[libre].type = FILE;
        SA[d_target].dir_entry[libre].size = SA[d_source].dir_entry[i].size;

        // se libera el entrada en el directorio original
        SA[d_source].dir_entry[i].used = false;

        return true;
End.
-----------------------------------------------------------------
```

Problema 3 (35 puntos)

Se desea modelar en ADA una fábrica de tanques de gasoil.

La misma posee diez puestos de fabricación, un planificador de maquinaria que determina el tipo de tanque a construir y un inspector de calidad.

Los puestos de fabricación fabrican simultáneamente el mismo tipo de tanque (lote) especificado por el planificador.

Cuando todos los puestos terminan de fabricar su tanque el planificador indicará el nuevo lote consistente del tipo de tanque a construir.

En cualquier momento el inspector de calidad podrá rechazar la calidad de uno de los tanques finalizados en el actual lote. En este caso el puesto que construyó el tanque defectuoso deberá fabricar uno nuevo por lo que el planificador no podrá indicar un nuevo lote hasta que el tanque sea rehecho.

Se deberá implementar en ADA las tareas: Planificador, Inspector de calidad y los 10 Puestos considerando que el Inspector de calidad solamente puede comunicarse con el Planificador.

Se dispone de las siguientes funciones:

- **definir_nuevo_lote(): tipo_lote** Ejecutado por el planificador retorna el tipo de lote.
- **construir(tipo_lote): tanque** Ejecutado por el puesto, retorna el tanque finalizado
- **inspeccionar(Lista(int)): int** Ejecutado por el Inspector, recibe una lista de puestos que han finalizado el tanque y retorna el número de puesto del tanque a rehacer o **-1** si están todos bien.
- **otras_tareas()** Ejecutado por el Inspectgor cuando no esta inspeccionando.

Nota:

- No se podrán utilizar tareas auxiliares.

Solución Problema 3:

```
task Planificador is
   entry finConstruccion(in idPuesto: Integer);
   entry comenzar_inspeccion(out finalizados: List of Integer);
   entry fin_inspeccion(in error: Integer);
end Planificador;
task body Planificador is
   lote: tipo_lote;
   terminados: List of Integer;
   error: integer;
begin
   for i in 1..9 loop
         puestos(i).identificador(i);
   end loop;
   terminados = crear_terminados(); -- crea la lista de terminados
   loop
         lote := definir_nuevo_lote();
         for i in 0..9 loop
               puesto(i).fabricar(lote);
         end loop;
         terminados.empty(); -- vacia la lista de terminados
         while terminados.size() < 10 loop
               select
                     when terminados.size() > 0;
                           accept comenzar_inspeccion(terminados);
                           accept fin_inspeccion(e) do
                                 error = e;
                           end fin_inspeccion;
                           if(error >= 0)
                                 terminados.remove(error);
                                 puesos(error).fabricar(lote);
                           end if;
                     or
                           accept finConstruccion(in idPuesto: Integer) do
                                 terminados.add(idPuesto);
                           end
               end select;
         end loop;
   end loop;
end Planificador;
task Inspector is
```

```
   end Inspector;

   task body Inspector is
      error: Integer;
      finalizados: Lista of Integer;
   begin
      loop
            Planificado.comenzar_inspeccion(finalizados);
            error := inspeccionar(finalizados);
            Planificado.fin_inspeccion(error);
            otras_tareas();
      end loop;
   end Inspector;

   task type Puesto  is
      entry identificador(in id: Integer);
      entry fabricar(in lote:tipo_lote)
   end Puesto;

   task body Puesto  is
      idPuesto: Integer;
      datosLote: tipo_lote;
   begin
      accept identificador(in id) do
            idPuesto := id;
      end identificador;
      loop
            accept fabricar (in lote:tipo_lote) do
                  datosLote := lote;
            end fabricar;
            construir(datosLote);
            Planificador.finConstruccion(idPuesto);
      end loop;
   end Puesto;

   puestos: array (0..9) of Puesto;
-----------------------------------------------------------------------
```