# Examen – 27 de febrero de 2016

**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.**
- **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.**
- **Al momento de finalizar el examen no se podrá escribir absolutamente nada en las hojas, debiéndose parar e ir a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración forma parte de la duración del examen.**

## Problema 1 ( 32 puntos)

1. Enumere y explique cuales son los tres requisitos que debe cumplir una solución al problema de la sección crítica.

2. Describa los sistemas de multiprocesadores simétricos y asimétricos e indique ventajas y desventajas.

3. Explique porqué debe protegerse el vector de interrupciones y las rutinas de atención a las interrupciones. ¿Cómo se logra esta protección mediante hardware?

4. Enumere y describa brevemente tres métodos de planificación de disco.

5. a. ¿Los manejadores de dispositivos (*device drivers*) son un componente de hardware o software?
   b. ¿Cuál es su función principal?

6. Describa brevemente los pasos para el manejo de un fallo de páginas (una vez que se determinó que éste ocurrió) en un modelo de paginación bajo demanda pura.

7. 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. Indique justificando brevemente:
   a) ¿ De que tamaño son las páginas ?
   b) ¿ De que tamaño son los marcos (frames) ?
   c) Si las entradas de la tabla de páginas son de 4 bytes y cada tabla ocupa una página, ¿ Cómo se distribuye la componente de 20 bits para direccionar las páginas ?

8. Describa el nivel 5 de RAID

## Problema 2 (34 puntos)

Sea un sistema de archivos FAT que cuenta con la siguiente organización en el disco:

| **Contenido** | Sector de arranque | Comienzo de la FAT | Comienzo de la región de datos |
|---|---|---|---|
| **Sector** | 0 | 1 | 33 |

El sector de arranque comienza en el sector 0, la FAT en el sector 1, y la región de datos en el sector 33. Este sistema cuenta con las siguientes estructuras de datos:

```
type sector = array [0..4095] of byte;

type entrada_dir = record
        usado : bit;                                  // entrada usada o no (1 bit)
        nom : array [0..19] of char;                  // nombre del elemento (20 bytes)
        ext : array [0..2] of char;                   // extensión del elemento (3 bytes)
        tipo : {0=ARCHIVO, 1=DIRECTORIO}; // tipo de elemento (1 bit)
        inicio : int;                                 // dirección de comienzo (4 bytes)
        tam : int;                                    // tamaño del archivo en bytes (4 bytes)
        reservado : array [0..5] bits;                // espacio reservado por el sistema (6 bits)
end; // 32 bytes

type fat = array [0..32767] of int; // estructura de la FAT (2^17 bytes)
var F : fat;
```

Notas generales:

- La variable F es global y siempre se encuentra cargada en memoria con el contenido de la FAT.
- Archivos y directorios son alojados en la región de datos usando al estructura de la FAT. El directorio raíz comienza en el primer sector de la región de datos.
- La FAT utiliza números físicos de sectores. El valor 0 indica un sector libre y el 1 indica el fin de un archivo o directorio.
- Toda operación debe dejar el sistema de archivos en un estado consistente. Pueden utilizar funciones auxiliares. La implementación debe ser eficiente.

Por otra parte se dispone de los siguientes procedimientos:

- Procedure leerSect(sec_num: int, Var buff: sector, Var ok: boolean)
  Lee de disco el sector **sec_num**, pasado como parámetro, y retorna el contenido leído en el parámetro de salida **buff**. En el parámetro **ok** se retorna el éxito de la ejecución de la operación.
- Procedure escrSect(sec_num : int, buff : sector, Var ok : boolean)
  Escribe en el sector **sec_num**, pasado como parámetro, la información que se encuentra en el parámetro **buff**. En el parámetro **ok** se retorna el éxito de la ejecución de la operación.
- Procedure parteCamino(camino: array of char, var base: array of char, var resto: array of char);
  Retorna la primera parte de la ruta en el parámetro base, y las restantes partes en resto.
  Por ejemplo: parteCamino('Users\sistoper\a.txt', 'Users', 'sistoper\a.txt')
  parteCamino('a.txt', 'a.txt', '')
- Procedure parteNombre(nomext: array of char, var nom: array of char, var ext: array of char);

Recibe un nombre completo de archivo o directorio, y retorna su nombre en nom por un lado y su extensión en ext por otro.
Por ejemplo: parteNombre('a.txt', 'a', 'txt')
parteNombre('sistoper', 'sistoper', '')

Se pide:

1. ¿Cuál es el tamaño máximo de almacenamiento (región de datos) que soporta este sistema? Justifique su respuesta.
2. ¿Cuál es el tamaño máximo que puede tener un archivo en este sistema? Justifique su respuesta.
3. Sin considerar el contenido de sus subdirectorios, ¿Cuál es la cantidad máxima de elementos que puede contener el directorio raíz? Justifique su respuesta.
4. Implemente la función buscar que busca un elemento dentro de un sector de disco asignado a un directorio. El cabezal de la función debe ser el siguiente:

   Procedure buscar(sect_num: int, nomext: array of char, tipo: bit, Var buff: array [0..4095] of byte, Var indice: int, Var ok: bool);

   El argumento sect_num indica el sector del disco, nomext es el nombre completo del elemento (incluida su extensión), y tipo indica si el elemento es de tipo archivo o directorio. En el argumento buff debe retornarse el contenido del sector, en indice la ubicación del elemento en caso de haberse encontrado, y en ok se debe retornar true si la operación fue ejecutada con éxito, o false en caso contrario.
5. Implemente la función eliminar que elimina un archivo de forma lógica. El cabezal de la función debe ser el siguiente:

   Procedure eliminar(cam_arch: array of char, Var ok: boolean);

   El argumento cam_arch es la ruta completa al archivo a eliminar. En el argumento ok se debe retornar true si la operación fue ejecutada con éxito, o false en caso contrario.
6. Describa cómo modificaría la implementación de la parte 5 si quisiera realizar un borrado físico.

**Solución**

**1.**

La FAT tiene 32768 (2^15) entradas (se tienen 32 sectores dedicados a la FAT, donde cada uno tiene un tamaño de 2^12 bytes. Por lo tanto la estructura de la FAT ocupa un total de 2^12*32 = 2^12*2^5 = 2^17 bytes. Cada entrada de la FAT ocupa 4 bytes, por lo tanto, se tienen a lo sumo 2^17/ 2^2 = 2^15 entradas en la FAT).

Entonces, se pueden tener a lo sumo 2^15 sectores en el disco para la región de datos, lo cual equivale a 2^15*2^12 bytes = 2^17 Kb = 2^7 Mb = 128 Mb.

Por otro lado, los sectores del 0 al 32 no pueden ser utilizados para datos, ya que estos corresponden físicamente a los sectores utilizados para la información de la propia FAT y el sector de arranque.

Entonces hay 132 kb que no se pueden contabilizar (33 sectores). Por lo tanto el tamaño máximo de almacenamiento es 128Mb – 132Kb.

**2.**

La cantidad máxima de sectores es a lo sumo 2^15, pero los primeros 34 sectores están reservados (sector de arranque + FAT + directorio raíz).

Por otro lado, en la entrada de directorio, el tamaño de archivo se almacena en un campo de 4 bytes.

Por lo tanto, el tamaño máximo en bytes de archivo soportado es el mínimo entre (2^15-34)*2^12 y 2^32 bytes, es decir 128Mb-136Kb

**3.**

Cada entrada de directorio ocupa 2^5 bytes y cada sector tiene tamaño 2^12 bytes, por lo tanto hay 2^7 entradas de directorio por sector.

El directorio raíz puede usar hasta 2^15-33 sectores para su contenido. La cantidad máxima de elementos se alcanza cuando el directorio raíz contiene archivos de tamaño 0 y/o directorios vacíos. En ese caso puede contener hasta (2^15-33)*2^7 elementos.

**4.**

```
Procedure buscar(sect_num: int, nomext: array of char, tipo: bit, Var buff: array
[0..4095] of byte, Var indice: int, Var ok: bool) {
  var dir_entries: array [0..127] of entrada_dir;
  var i: int;

  leerSect(sect_num, buff, ok);
  if (!ok) return;
  dir_entries = (array [0..127] of entrada_dir)buff;
  indice = -1
  parteNombre(nomext, nom, ext);

  for(i=0; i < 128 && indice == -1; i++) {
    if (dir_entries[i].tipo == tipo && dir_entries[i].nombre == nombre &&
      dir_entries[i].ext == ext && dir_entries[i].usado) {
      indice = i;
    }
  }
}
```

**5.**

```
Procedure eliminar(cam_arch: array of char, Var ok: bool) {
  var buff: array [0..4095] of byte;
  var dir_entries: array [0..127] of entrada_dir;
  var indice: int;
```

```
  var sect: int;
  var tipo: int;
  var nomext: array of char;

  ok = true;
  sect = 33;
  indice = -1;

  while (ok) {
    parteCamino(cam_arch, nomext, cam_arch);

    if (cam_arch == '') tipo = 0;
    else tipo = 1;

    while (indice == -1 && sect != 1) {
      buscar(sect, nomext, tipo, buff, indice, ok);
      if (!ok) return;
      if (indice == -1) sect = F[sect];
    }

    if (indice == -1) {
      ok = false;
    } else {
      dir_entries = (array [0..127] of entrada_dir)buff;
      if (tipo == 0) {
        dir_entries[indice].usado = 0;
        escrSect(sect, buff, ok);
        if (ok) liberarFAT(dir_entries[indice].inicio, ok);
        return;
      } else {
        sect = dir_entries[indice].inicio;
        indice = -1;
      }
    }
  }
}

Procedure liberarFAT(sect: int, var ok:bool) {
  Var sig: int;
  var sectores: array [0..31] of boolean;
  for(sig=0; sig < 32; sig++)sectores[sig]=false;

  actual = sect;
  while (actual !=1) {
    sectores[actual/1024] = true;
    sig = F[actual];
    F[actual] = 0;
    actual = sig;
  }
  for(sig=0; sig < 32 && ok; sig++)
    if(sectores[sig] == true){
      escrSect(actual, F+(1024*sig), ok)
    }
}
```

**6.** Basta con modificar la función liberarFAT para sobreescribir cada sector del archivo antes de liberar el sector de la FAT.

Al comienzo del procedimiento inicializar un buffer (**buff**) de 4096 bytes con ceros, y en el bucle que marca los índices de la FAT como libres además se manda a escribir el sector asociado con el buffer:

```
escrSect(actual, buff, ok);
```

## Problema 3 (34 puntos)

Se desea modelar usando ADA la puerta de entrada de un hospital universitario. El hospital cuenta con 10 médicos y 30 boxes de atención con capacidad para un paciente cada uno. Además hay 50 estudiantes que diagnostican a los pacientes y un portero que dirige a los pacientes que llegan a un box libre si hay alguno o al box con menor cantidad de pacientes esperando en caso contrario.

Los estudiantes seleccionan un box ocupado para diagnosticar al paciente que allí se encuentra. Ellos anotan el diagnóstico en una planilla que luego será verificada por uno de los médicos. En un box determinado no podrá haber más de tres estudiantes en forma simultánea.

Los médicos seleccionan un box ocupado para atender al paciente. Deben esperar a que todos los estudiantes liberen el box antes de entrar y tienen prioridad sobre nuevos estudiantes que quieran entrar al box. Tampoco pueden entrar nuevos estudiantes mientras el médico está trabajando en el box. Cuando el médico termina el paciente se retira del hospital.

Si un paciente espera dentro del box por 20 minutos sin ser atendido por el médico, presiona el botón de queja por demora, y continuará presionandolo cada 20 minutos hasta ser atendido.

Se pide implementar las tareas: paciente, médico, estudiante, portero y box.

No se permiten tareas auxiliares.

Se dispone de las siguientes funciones:

| | |
|---|---|
| presionar_boton() | Ejecutada por el paciente para quejarse por la demora. |
| atender_paciente() | Ejecutada por los médicos para diagnosticar al paciente y revisar los informes de los estudiantes si los hubiere. |
| elegir_box(): integer | Ejecutada por los estudiantes y los médico para seleccionar un box ocupado. Si no hay ninguno ocupado se bloquea hasta que se ocupe alguno. |
| diagnosticar_paciente() | Ejecutada por los estudiantes para diagnosticar al paciente y anotarlo en la planilla. |
| tiempo(): integer | Retorna la cantidad de segundos desde el 1/1/2016. |

```
task portero is
      entry llega_paciente(box: out integer);
      entry se_va_paciente(box: in integer);
end

task body portero is

var boxes: array[1..30] of integer;
var nbox, tbox: integer
begin
```

```
	for i:= 1 to 10 do
		doctores[i].obtener_id(i);
	end;
	nbox := 1;
	loop
		select
			accept llega_paciente(box: out integer) do
				box = nbox;
			end accept
			boxes[nbox] = boxes[nbox] + 1;
			for int i:= 2 to 30 do
				if boxes[i] < boxes[nbox] then
					nbox := i;
				end if
			end for
		or
			accept se_va_paciente(box: int integer)
				tbox = box;
			end accept
			boxes[tbox] = boxes[tbox] - 1;
			if(tbox <> nbox AND boxes[tbox] < boxes[nbox])
				nbox = tbox;
			end if
		end select
	end loop
end


task type paciente is

var box: integer;
    atendido: boolean;
begin
	portero.llega_paciente(box);
	boxes[box].hay_paciente();
	boxes[box].entra_paciente(atendido);
	while not atendido do
		presionar_boton();
		boxes[box].entra_paciente(atendido);
	end
	portero.se_va_paciente(box);
end


task type estudiante is

var box: integer;
begin
	loop
		box := elegir_box();
		boxes[box].entra_estudiante();
		diagnosticar_paciente();
		boxes[box].sale_estudiante();
	end loop
end


task type doctor is
	entry obtener_id(id: in integer);
	entry atender();
end

task body doctor is

var box: integer;
    miId: integer;
begin
	accept obtener_id(id: in integer) do
```

```
            miId = id;
      end
      loop
            box := elegir_box();
            boxes[box].entra_doctor(miId);
            accept atender() do
                  atender_paciente(miId);
            end;
      end loop
end


task type box is
      entry entra_estudiante();
      entry sale_estudiante();
      entry entra_doctor(id: in integer);
      entry entra_paciente(ok: out boolean);
end

task body box is

var estudiantes: integer;
    espera, llegada: integer;
    fin, paciente: boolean;
begin
      estudiantes := 0;
      paciente := false;
      loop
            select
                  when not paciente =>
                  accept hay_paciente;
                  paciente := true;
            or
                  accept entra_paciente(ok: out boolean) do
                        llegada := tiempo();
                        espera := 20 * 60;
                        fin := false;
                        while not fin do
                              select
                                    when (estudiantes < 3) and
                                                      (entra_doctor'count = 0) =>
                                    accept entra_estudiante;
                                    estudiantes++;
                              or
                                    accept sale_estudiante;
                                    estudiantes--;
                              or
                                    when estudiantes = 0 =>
                                    accept entra_doctor(id: in integer);
                                    doctores[id].atender();
                                    fin := true;
                                    ok := true;
                                    paciente := false;
                              or
                                    delay espera;
                                    fin := true
                                    ok: = false;
                              end select;
                              espera := espera - (tiempo() - llegada);
                        end while
                  end accept
            end select
      end loop
end

doctores: array [1..10] of task doctor;
estudiantes: array[1..50] of task estudiante;
boxes: array [1..30] of task box;
```