Examen 14 de febrero de 2018

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)

Conteste explicando y justificando brevemente cada una de las siguientes preguntas:

1. Describa ventajas y desventajas de manejo de memoria a través de segmentación en comparación con manejo por páginación.
2. En el contexto de virtualización:
   a) Describa y compare hipervisores tipo I y II
   b) Explique el mecanismo de traducción binaria e indique en qué tipos de hipervisor puede ser utilizado.
3. Describa dos mecanismos de protección que le brinda el hardware al sistema operativo.
4. Describa los estados de un proceso y realice un diagrama mostrando las transiciones de un estado a otro, mencionando los eventos que pueden producir el cambio de estado.
5. Describa el problema de hiperpaginación (thrashing) y un método para evitarlo.
6. Describa tres métodos de planificación de disco
7. Compare los métodos de E/S programada y por interrupciones. Indique ventajas y desventajas de cada método.
8. Describa qué entiende por fragmentación interna y fragmentación externa.

Problema 2 (33 puntos, 5/9/9/10)

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

```
const MAX_SECTORS = 65536;
type sector = array [0..4095] of byte;
type entrada_dir = Record
            usado : bit;                    // entrada usada o no (1 bit)
            nom : array [0..22] of char;    // nombre del elemento (23 bytes)
            inicio : integer;               // dirección de comienzo (4 bytes)
            tipo : {ARCHIVO, DIRECTORIO};   // tipo de elemento (1 bit)
            tam : integer;                  // tamaño del archivo en bytes (4 bytes)
            reservado : array[0..5] of bit; // espacio reservado por el sistema (6 bits)
      end; // 32 bytes
type fat = array [0..(MAX_SECTORS-1)] of -2..(MAX_SECTORS-1);
type disk = Array [0..(MAX_SECTORS-1)] of sector;
Var   F : fat;
      D : disk;
type ubicacion_entrada = Record
            entrada : entrada_dir;
            sector : integer;
            indice : integer; // Indice de la entrada de directorio dentro del sector
      end;
```

Notas generales:

- La variable F es global y siempre se encuentra cargada en memoria con el contenido de la FAT. **No** se debe modelar la persistencia en disco de la FAT que además se sabe que se encuentra fuera del rango 0..MAX_SECTORS.
- Archivos y directorios son alojados en D usando la estructura de la FAT. El directorio raíz comienza en el sector número 0 de D.
- En la FAT el valor -1 representa "fin de archivo" y el -2 "sector libre".

Por otra parte se dispone de los siguientes procedimientos:

- `Procedure leerSect(sec_num: int, Var buff: sector) : boolean`
  Lee de disco el sector **sec_num**, pasado como parámetro, y carga el contenido leído en el parámetro de salida **buff**. Retorna el éxito de la ejecución de la operación.
- `Procedure escrSect(sec_num : int, buff : sector) : boolean`
  Escribe en el sector **sec_num**, pasado como parámetro, la información que se encuentra en el parámetro **buff**. 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', '')
- `Function armaCamino(camino: array of char, elemento: array of char): array of char;`
  Retorna la un camino conformado por el camino y el elemento
  Por ejemplo: armaCamino('Users\sistoper', 'a.txt') retorna 'Users\sistoper\a.txt'

Se pide:

1. ¿Cuál es la cantidad máxima de elementos que puede contener el directorio raíz ?. Justifique.
2. Implementar una función que busca un elemento dentro del sistema de archivos:
   `Function buscar(cam: array of char): ubicacion_entrada;`
   Donde `cam` es la ruta completa al elemento que se desea buscar. La función retorna la entrada y ubicación del elemento si fue encontrado o NULL si no se encontró el elemento o se produjo un error.
3. Implementar una función que busca/crea una entrada libre en un directorio:
   `Function entradaLibre(dir: entrada_dir): ubicacion_entrada;`
   Donde `entrada_dir` es la entrada de directorio donde se desea buscar una entrada libre. La función retorna una entrada y ubicación donde la entrada esta libre o NULL si no hay ninguna libre y no se puede crear una nueva o se produjo un error.
4. Implementar una función que mueva un archivo con la siguiente definición:
   `Function mover(camOrigen: array of char, camDestino : array_of_char): boolean;`
   Donde `camOrigen` es la ruta completa al archivo que se desea mover y `camDestino` es la ruta completa al directorio destino. Retorna un booleano indicando si la operación fue realizada con éxito. Esta operación no mueve directorios ni crea directorios.

Solucion:

1) La cantidad máxima de elementos en el directorio raíz se da cuando todos los bloques de datos son utilizados para almacenar información de dicho directorio, conteniendo en todas las entradas archivos de tamaño 0 que no requieren bloques de datos.
Entonces tenemos que:
Cada sector de disco tiene 4096 bytes = 2^12 bytes, y cada entrada ocupa 32 bytes = 2^5 bytes.
Por lo tanto, cada sector puede contener 128 entradas = 2^7 entradas
Multiplicamos ese valor por la cantidad de sectores de datos (65536=2^16) y llegamos a 2^23 elementos.

2)

```
function buscar(cam: array of char): ubicacion_entrada
var
  sector, i: int;
  elem: array of char;
  entradas: array[0..127] of entrada_dir;
  ret: ubicacion_entrada;
{
  if(cam == "")
      return NULL;
  sector = 0; // Comienzo de la raiz
  parteCamino(cam, elem, cam);
  while(true)
  {
      if(!leerSect(sector, entradas))
        return NULL;
      for(i = 0; i < 128; i++)
      {
            if(entradas[i].usado &&
               entradas[i].nom == elem)
            { // Encontre la entrada
              if(cam == "")
              { // Encontre el elemento
                ret.entrada = entradas[i]
                ret.sector = sector;
                ret.indice = i;
                return ret;
              }
              if(entradas[i].tipo == ARCHIVO)
                return NULL;
              sector = entradas[i].inicio;
              parteCamino(cam, elem, cam);
              break;
            }
            else if(i == 127)
            { // ultima entrada de este sector
              sector = F[sector];
              if(sector < 0) // llegue al final y no encontre el elemento
                return NULL;
            }
      }
  }
}
```

3)

```
function entradaLibre(dir : entrada_dir): ubicacion_entrada
var
  sector, previo, i: int;
  entradas: array[0..127] of entrada_dir;
  ret: ubicacion_entrada;
{
  if(dir == NULL || dir.tipo == ARCHIVO || !dir.usado)
    return NULL;
  previo = -1;
  sector = entrada_dir.inicio;
  while(sector != -1)
  {
      if(!leerSect(sector, entradas))
        return NULL;
      for(i = 0; i < 128; i++)
```

```
      {
            if(!entradas[i].usado)
                  break;
      }
      if(i < 128)break;
      previo = sector;
      sector = F[sector];
  }

  if(sector == -1)
  {
      for(sector = MAX_SECTORS - 1; sector >= 0; sector--)
            if(FAT[sector] == -2)
                  break; // encontre sector libre

      if(sector == -1)
        return NULL;  // no encontre sector libre
      for(i = 0; i < 127; i++)
        entradas[i].usado = false;
      if(!escrSect(sector, entradas)) // Inicializo sector
        return NULL;
      i = 0;
      if(previo == -1)
        entrada_dir.inicio = sector;
      else
        FAT[previo] = sector;
      FAT[sector] = -1;
  }
  ret.entrada = entradas[i];
  ret.sector = sector;
  ret.indice = i;
  return ret;
}

4)
function mover(camOrigen: array of char, camDestino : array_of_char): boolean
var
  origen, destino: ubicacion_entrada;
  entradas: array[0..127] of entrada_dir;
{
      origen = buscar(camOrigen);
      if(origen == NULL || origen.entrada.tipo == DIRECTORIO)
            return false; // No existe origen o es un directorio
      destino = buscar(camDestino);
      if(destino == NULL || destino.entrada.tipo == ARCHIVO)
            return false; // No existe destino o es un archivo
      if(buscar(armaCamino(camDestino, origen.nom)) != NULL)
            return false; // Ya existe el archivo en el directorio destino
      destino = entradaLibre(destino.entrada);
      if(destino == NULL)
            return false;
      destino.entrada = origen.entrada; // Copio datos de la entrada
      if(!leerSect(destino.sector, entradas))
        return false;
      entradas[destino.indice] = destino.entrada;
      if(!escrSect(destino.sector, entradas) ||
         !leerSect(origen.sector, entradas))
            return false;
      entradas[origen.indice].usado = false;
      return escrSect(origen.sector, entradas);
}
```

Problema 3 (35 puntos)

Se desea modelar el control de entrada a una tribuna de un estadio. La tribuna cuenta con 6 molinetes de entrada en los cuales se valida el ticket de los espectadores. Al llegar los espectadores deben elegir el molinete con la cola más corta. Si la validación es exitosa el espectador entra al estadio y sino debe irse.

Además hay un supervisor que recoge los talones de los tickets cada vez que pasaron 500 espectadores entre todos los molinetes. El molinete que deja pasar al espectador número 500 debe avisarle al supervisor y el resto de los molinetes no deben dejar pasar nuevos espectadores hasta que el supervisor termine (pueden validar el ticket del espectador que esta frente al molinete pero no dejarlo continuar).

También hay un auditor externo que revisa los tickets periódicamente. El auditor no puede auditar si hay algún molinete validando o el supervisor trabajando, y además tiene la máxima prioridad.

Se pide: implementar usando mailboxes los procesos espectador, molinete, auditor y supervisor. No se permiten tareas auxiliares. Se debe especificar la semántica de los mailboxes utilizados teniendo en cuenta que solo disponen de las operaciones send y receive.

Se dispone de las siguientes funciones auxiliares:

- validar_ticket(ticket): boolean — Invocada por el molinete para validar el ticket
- mi_ticket(): ticket — Invocada por el espectador para obtener su ticket
- entrar() — Invocada por el espectador para entrar
- irse() — Invocada por el espectador para irse
- recoger_talones() — Invocada por el supervisor para recoger los talones de todos los molinetes
- auditar() — Invocada por el auditor para hacer su trabajo
- procesar_auditoría() — Invocada por el auditor luego de auditar

Solución:

Se utilizan mailboxes infinitos con resolución FIFO, send no bloqueante, receive bloqueante.

```
VAR
 mb_colas: Mailbox of array[1..6] of integer;
 mb_cant: Mailbox of integer;
 mb_molinete_espero: array[1..6] of Mailbox of Nil;
 mb_molinete_ticket: array[1..6] of Mailbox of ticket;
 mb_molinete_permiso: array[1..6] of Mailbox of Boolean;
 mb_molinete_ack: array[1..6] of Mailbox of Nil;
 mb_supervisor: Mailbox of Nil;
 mb_auditor_ocupado: Mailbox of Nil;
 mb_prio_auditor: Mailbox of Nil;
 mb_supervisado: Mailbox of Nil;
```

```
procedure auditor()
VAR i:integer;
begin
  while(true)
  {
    receive(mb_auditor_ocupado);
    for(i=1 to 6)
      receive(mb_prio_auditor);
    auditar();
    send(mb_auditor_ocupado, nil);
    for(i=1 to 6)
      send(mb_prio_auditor, nil);
    procesar_auditoria();
  }
end

procedure supervisor()
begin
  while(true)
  {
    receive(mb_supervisor);
    recoger_talones();
    send(mb_supervisado, nil);
    send(mb_cant, 0);
    send(mb_auditor_ocupado, nil);
  }
end

procedure espectador()
VAR
  colas: array[1..6] of integer;
  i, cola, cant: integer;
  ticket: ticket;
  pasar: boolean;
begin
  ticket = mi_ticket();
  colas = receive(mb_colas);
  cant = colas[1]; cola = 1;
  for(i = 2; i < 6; i++)
  {
    if(cant < colas[i])
    {
      cant = colas[i];
      cola = i;
    }
  }
  colas[cola]=colas[cola]+1;
  send(mb_colas, colas);

  mb_molinete_espero[cola].receive();
  mb_molinete_ticket[cola].send(ticket);
  pasar = mb_molinete_permiso[cola].receive();
  mb_molinete_ack[cola].send(nil);

  colas = receive(mb_colas);
  colas[cola]=colas[cola]-1;
  send(mb_colas, colas);
  if(pasar)
    entrar();
  else irse();
end
```

```
procedure molinete(id: integer)
VAR
  cant: integer;
  ticket: ticket;
  pasar: boolean;
begin
  while(true)
  {
    mb_molinete_espero[id].send(nil);
    ticket = receive(mb_molinete_ticket[id]);
    receive(mb_prio_auditor);
    pasar = validar_ticket(ticket);
    send(mb_prio_auditor, nil);
    cant = receive(mb_cant);
    if(pasar)
    {
      cant = cant + 1;
      if(cant == 500)
      {
        receive(mb_auditor_ocupado);
        send(mb_supervisor, nil);
        receive(mb_supervisado);
      }
      else
       send(mb_cant, cant);
    }else send(mb_cant, cant);
    mb_molinete_permiso[id].send(pasar);
    mb_molinete_ack[id].receive();
  }
end;

// Programa principal
begin
VAR colas: array[1..6] of integer;
begin
  for i = 1 to 6
  {
    colas[i] = 0;
    send(mb_prio_auditor, nil);
  }
  send(mb_colas, colas);
  send(mb_cant, 0);
  send(mb_auditor_ocupado, nil);
  cobegin
      auditor();
      supervisor();
      molinete(1);
      molinete(2);
      molinete(3);
      molinete(4);
      molinete(5);
      molinete(6);
      espectador();
      ...
      espectador();
  coend
end
// Esta solución cuenta la cantidad de espectadores cuyo ticket es válido.
También era válido contar además los espectadores con ticket inválido.
```