Segundo Parcial Julio 2016

Lea detenidamente las siguientes instrucciones. No cumplir los requerimientos puede implicar la pérdida de los puntos del parcial.

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 según el orden de hojas.**

Dudas

- **Sólo se contestarán dudas de letra.**
- **No se aceptarán dudas en los últimos 15 minutos del parcial.**

Material

- **El examen es SIN material (no puede utilizarse ningún apunte, dispositivo móvil, libro ni calculadora). Sólo puede tenerse las hojas del parcial, lápiz, goma y lapicera en su banco. Todas sus demás pertenencias debe colocarlas en el piso debajo de su asiento.**

Finalización

- **El parcial dura 3 horas 15 minutos.**
- **Al momento de finalizar el parcial 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 parcial.**

Problema 1 (10 puntos)

1. Indique los métodos de asociación de direcciones (*address binding*) y mencione el soporte de hardware necesario para su implementación.
2. Nombre 3 tipos de *malware* (software malicioso) y explique muy brevemente cada uno de ellos.
3. Describa la estructura de tabla de páginas invertida indicando ventajas y desventajas respecto a la estructura de tabla de páginas jerárquica.
4. En el contexto de E/S, describa y compare las operaciones bloqueantes y no bloqueantes.
5. Describa qué problemas enfrenta un hipervisor para poder soportar máquinas virtuales con sistemas operativos invitados que tienen soporte de memoria virtual y cómo puede resolverlos.

Problema 2 (23 puntos) (5,6,6,6)

Se tiene un sistema de archivos FAT que cuenta con las siguientes estructura de datos.

```
type sector = array [0..511] of byte;
type fs = array [0..(MAX_DIRS-1)] of dir;
type dir = Record
   dir_entry : array [0..(MAX_ENTRIES_DIR-1)] of entry;
   usado : boolean; // marca si la entrada está usada
end;
type entry = Record
   usado : boolean;                // marca si la entrada está usada
   tipo : {ARCHIVO, DIRECTORIO};   // tipo de entrada
   nombre : array [0..7] of char;  // nombre del archivo o directorio
   ext : array [0..2] of char;     // extensión del archivo
   comienzo : integer;             // dirección de inicio del archivo
   tamaño : integer;               // tamaño del archivo en bytes
   fs_indice : 1..(MAX_DIRS-1);    // índice en fs con contenido del directorio
end;
type fat = array [0..(MAX_SECTORS_DISK-1)] of -1..(MAX_SECTORS_DISK-1);
var  SA : fs;     // sistema de archivos
     F  : fat;    // tabla de asignación de archivos
```

Notas generales:

- Las variables `F` y `SA` son globales y se encuentran siempre en memoria. `SA` contiene el árbol de archivos y directorios, 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 `fs_indice` es usado solamente cuando el entry es tipo directorio.
- El campo `comienzo` es usado solamente cuando el entry es tipo archivo.
- Toda operación debe dejar el sistema de archivos en un estado consistente.

Se pide:

1. ¿Cuál es la cantidad máxima de archivos y directorios que pueden existir en este sistema?
2. Implementar una función que verifique que la correctitud de un archivo de forma de que no contenga ciclos y tenga su marca de finalización correspondiente.

   ```
   Function checkArchivo(archivo: entry): boolean;
   ```

   Donde `archivo` es el entry del archivo a verificar. La operación debe retornar si finalizó con éxito.
3. Implementar una función que verifique todos los archivos del sistema siguiendo los criterios de la parte 2 y además verificando que dos archivos no puedan compartir un mismo bloque. La operación debe retornar si finalizó con éxito.

   ```
   Function checkFS(): boolean;
   ```

4. Implementar una función que mueva un archivo.

   ```
   Function moverArchivo(d_origen: entry; nombre: array [0..7] of char; ext:
   array [0..2] of char; d_destino: entry): boolean;
   ```

   Donde `d_origen` es el entry del directorio origen, `nombre` y `ext` es el nombre y extensión del archivo origen, `d_destino` es el entry del directorio destino. La operación debe retornar si finalizó con éxito.

Solución

1)

Cantidad máxima de directorios: MAX_DIRS
Dado que todos los directorios están mapeados al arreglo FS, el largo del arreglo limita el máximo de directorios.
Cantidad máxima de archivos: MAX_DIRS * MAX_ENTRIES_ON_DIR - MAX_DIRS + 1
Si consideramos sólo archivos vacíos, la tabla FAT no limita la cantidad ya que no se utilizaría. Por lo tanto, el límite de elementos que puede contener el sistema de archivos está dado por la cantidad de directorios multiplicado por la cantidad de elementos que entran por directorio.
Es decir que la cantidad máxima de elementos del sistema de archivo es MAX_DIRS * MAX_DIR_ENTRIES_ON_DIR.
Como los directorios (salvo la raíz) deben ser referenciados desde el directorio "padre" en el sistema de archivos, ocupan una entrada en su padre por lo que quitan un lugar de un posible archivo.
Por lo tanto, la cantidad máxima es MAX_DIRS * MAX_ENTRIES_ON_DIR - MAX_DIRS + 1.

2)

```
function checkArchivo( archivo: entry ): boolean {
      var ind_usados : array[0..(MAX_SECTORS_DISK-1)] of boolean;
      if ( !archivo.usado or archivo.tipo != ARCHIVO ) return False;

      int indice = archivo.comienzo;
      int bloques = ceil(archivo.tamaño / 512.0);
      while ( indice > 0 and indice < MAX_SECTORS_DISK ) {
            if ( indices_usados[ indice ] or bloques <= 0 )
                  return False;
            indices_usados[ indice ] = True;
            bloques--;
            indice = F[ indice ];
      }
      return (bloques == 0 && indice == 0);
}
```

3)

```
function checkFS(): boolean {
      var ok : boolean;
      var ind_usados : array[0..( MAX_SECTORS_ON_DISK-1)] of boolean;
      var actual : entry;
      var lista_dir : array [0..(MAX_DIRS-1)] of integer;
      var sig_dir, tope_dir, i : integer;

      tope_dir = 1;
      sig_dir = 0;
      lista_dir[0] = 0; // raiz

      // Chequeo todos los archivos del FS
      while ( sig_dir < tope_dir ) {
            for ( i=0; i < MAX_ENTRIES_DIR; i++ ) {
                  actual = SA[lista_dir[sig_dir]].dir_entry[i];
                  if ( actual.usado ) {
                        if ( entry.type == ARCHIVO ) {
                              if ( !checkFile( actual ) ||
                              ind_usados[actual.comienzo]) {
                                    return False;
                              } else {
                                    ind_usados[actual.comienzo] = True;
                              }
```

```
                        } else if ( entry.type == DIRECTORIO ) {
                              lista_dir[tope_dir] = actual.fs_indice;
                              tope_dir++;
                        }
                  }
            }
            sig_dir++;
      }
      // Chequeo que no se compartan bloques
      for ( i=1; i < MAX_SECTORS_ON_DISK; i++ ) {
            if ( F[i] > 0 ) {
                  if ( ind_usados[F[i]] ) {
                        return False;
                  } else {
                        ind_usados[F[i]] = True;
                  }
            }
      }
      return True;
}
```

4)

```
function moverArchivo( d_origen: entry; nombre: array [0..7] of char; ext: array [0..2] of
char; d_destino: entry ): boolean {
      var i, j, libre : integer;
      var entry_origen : entry;
      var entry_destino : entry;

      if ( !d_origen.usado or !d_destino.usado) return False;
      if ( d_origen.tipo != DIRECTORIO or d_destino.tipo != DIRECTORIO) return False;

      // busco entry origen
      for ( i = 0; i < MAX_ENTRIES_DIR; i++) {
            entry_origen = SA[d_origen.fs_indice].dir_entry[i];

            if ( entry_origen.used and entry_origen.tipo == ARCHIVO
                  and entry_origen.name == nombre and entry_origen.ext == ext ) {

                  libre = -1;

                  // busco entry destino
                  for ( j = 0; j < MAX_ENTRIES_DIR; j++) {
                        entry_destino = SA[d_destino.fs_indice].dir_entry[j];

                        if ( entry_destino.used ) {
                              if ( entry_destino.tipo == ARCHIVO
                                    and entry_destino.name == nombre
                                    and entry_destino.ext == ext ) {
                                    return False;
                              }
                        } else if ( libre == -1 ){
                              libre = j;
                        }
                  }

                  if ( libre >= 0 ) {
                        entry_destino = SA[d_destino.fs_indice].dir_entry[libre];
                        entry_destino.usado     = True;
                        entry_destino.tipo      = entryOrigen.tipo ;
                        entry_destino.nombre    = entryOrigen.nombre;
                        entry_destino.ext       = entryOrigen.ext;
                        entry_destino.comienzo  = entryOrigen.comienzo;
                        entry_destino.tamano    = entryOrigen.tamano;
                        entry_destino.fs_indice = entryOrigen.fs_indice ;
                        entry_origen.usado      = False;
```

```
                        return True;
                    } else {
                        return False;
                    }
                }
        }
        return False;
}
```

Problema 3 (30 puntos)

Se desea modelar un depósito de almacenamiento de productos que tiene 100 estanterías.

En el depósito trabajan 7 empaquetadores los cuales reciben hojas de pedidos de varios productos y deben obtenerlos todos juntos para completar el pedido. Antes de retirarlos de las estanterías deberán verificar que hay stock de todos los productos y en caso de que falte algún producto deberán indicar que el pedido no puede completarse.

También hay 4 inspectores de mercadería que inspeccionan la mercadería de las estanterías. Puede haber varios inspectores de mercadería trabajando simultáneamente en la misma estantería.

Los empaquetadores necesitan el uso exclusivo de las estanterías que están usando.

Los inspectores tienen prioridad sobre los empaquetadores para el acceso a la estantería.

Se pide: implementar las tareas empaquetador e inspector usando monitores.

No se permite usar tareas auxiliares.

Se cuenta con las siguientes funciones auxiliares:

- obtener_pedido(): array [1..20] of integer
  - Retorna un arreglo con los productos del pedido. El primer NULL indica el fin de la lista.
- donde_esta(producto: integer): integer
  - Retorna el número de estantería donde se almacena el producto. Cada producto solo puede estar en una estantería.
- hay_stock(producto: integer, estantería: integer): boolean
  - Indica si hay stock del producto en la estantería
- retirar(producto: integer, estantería: integer)
  - Saca el producto de la estantería y lo pone en la caja
- finalizar_pedido(ok: boolean)
  - Si ok cierra la caja y la envía a expedición, en otro caso indica que el pedido no puede completarse
- donde_inspeccionar(): integer
  - Indica la estantería donde el inspector de mercadería debe inspeccionar
- inspeccionar(estantería: integer)
  - Inspecciona la mercadería de la estantería

Solución:

```
type Monitor Estanteria
{
  Condition inspector, empaquetador, retirar;
  bool ocupado = false, a_retirar = false;
  int inspectores = 0;
  void inspeccionar()
  {
    if(inspectores == 0 && a_retirar)
      inspector.Wait(); // Si empaquetador esta retirando, espero
    inspectores = inspectores + 1;
    inspector.Signal();
  }

  void fin_inspeccionar()
  {
    inspectores = inspectores - 1;
    if(inspectores == 0)
      retirar.Signal();
    }
  }

  void obtener()
  {
    if(ocupado)
      empaquetador.Wait();
    ocupado = true;
  }

  void voy_a_retirar()
  {
    a_retirar = true;
    if(inspectores > 0)
      retirar.Wait();  // Si hay inspectores, espero para poder retirar
  }

  void liberar()
  {
    ocupado = false;
    a_retirar = false;
    inspector.Signal();
    empaquetador.Signal();
  }
}

var: Estanterias: array[1..100] of Estanteria;

main()
{
  cobegin
    empaquetador(); empaquetador(); empaquetador(); empaquetador();
    empaquetador(); empaquetador(); empaquetador();
    inspector(); inspector(); inspector(); inspector();
  coend
}
```

```
empaquetador()
  var:
     pedido: array [1..20] of integer;
     donde: array [1..20] of integer;
     usado: array [1..100] of boolean;
     i,tope: integer;
     ok: boolean;
{
  while(true)
  {
    pedido = obtener_pedido();
    ok = true;
    for(i = 1; pedido[i] <> NULL && i <= 20; i++)
    {
      donde[i] = donde_esta(pedido[i]);
      usado[donde[i]] = true;
    }
    tope = i;

    // Obtengo todas las estanterias usadas y verifico stock
    // Pido ordenado por número de estantería para evitar deadlock
    for(i = 1; i <= 100 && ok; i++)
    {
      if(usado[i])
      { // "Lockeo" estanteria (uso exclusivo entre empaquetadores)
        Estanterias[i].obtener();
        for(j = 1; j < tope; j++)
        { // verifico stock del los productos de esa estantería
          if(donde[j] = i && !hay_stock(pedido[j],i))
            ok = false;
        }
      }
    }

    for(; i >= 0; i--)
    {
      if(usado[i])
      {
        if(ok)
        {
          Estanterias[i].voy_a_retirar(); // Además exclusivo con inspectores
          for(j = 1; j < tope; j++)
          {
            if(donde[j] = i) // retiro producto
              retirar(pedido[j], i);
          }
        }
        Estanterias[i].liberar(); // Libero todos los locks
      }
    }
    finalizar_pedido(ok);
  }
}

inspector()
{
  while(true)
  {
    int donde = donde_inspeccionar();
    Estanterias[donde].inspeccionar();
    inspeccionar(donde);
```

```
    Estanterias[done].fin_inspeccionar();
  }
}
```

Otra solución

```
Monitor Estanterias {
      Condition [100] inspector
      Condition [7] empaquetadores;
      int [7][20] pedidos;
      bool [100] ocupado
      int [100] inspectores;

      void inspeccionar(int est) {
            inspectores[est] = inspectores[est] + 1;
            if(ocupado[est])
                  inspector[est].Wait();
            inspector[est].Signal();
      }

      void fin_inspeccionar(int est) {
            inspectores[est] = inspectores[est] - 1;
            if(inspectores[est] = 0) {
                  liberarEmpaquetadores();
            }
      }

      void obtener(int [20] estanterias, int id) {
            for (int i=0; i<20 && estanterias[i] != null; i++) {
                  pedidos[id][i] = estanterias[i];
            }
            if(!hayLugar(id) && pedidos[id][0] >= 0)
                  empaquetadores[id].Wait();
            else
               for (int i=0; i<20 && pedidos[id][i] >= 0; i++){
                  ocupado[pedidos[id][i]] = true;
               }
      }

      void liberar(int id) {
            for (int i=0; i<20 && pedidos[id][i] >= 0; i++) {
                  ocupado[pedidos[id][i]] = false;
                  if(inspectores[pedidos[id][i]] > 0)
                        inspector[pedidos[id][i]].Signal();
                  pedidos[id][i] = -1;
            }
            liberarEmpaquetadores();
      }

      boolean hayLugar(int id) {
            if(pedidos[i][0] == -1)
                  return false;
            for (int i=0; i<20 && pedidos[id][i] >= 0; i++) {
                  if (ocupado[pedidos[id][i]] || inspectores[pedidos[id][i]]>0)
                        return false;
            }
            return true;
      }

      void liberarEmpaquetadores() {
            for (int i=0; i<7; i++) {
```

```
                  if (hayLugar[i]) {
                        for (int j=0; j<20 && pedidos[i][j] >= 0; j++) {
                              ocupado[pedidos[i][j]] = true;
                        }
                        empaquetadores[i].Signal();
                  }
            }
      }

      begin  // Inicializo variables
            for (int i=0; i<7; i++) {
                  for (int j=0; i<20; i++)
                        pedidos[i][j] = -1;
            }

            for (int i=0; i<100; i++)
                  ocupados[i] = false;
      end
}

main()
{
  cobegin
    empaquetador(0); empaquetador(1); empaquetador(2); empaquetador(3);
    empaquetador(4); empaquetador(5); empaquetador(6);
    inspector(); inspector(); inspector(); inspector();
  coend
}

empaquetador(int id) {
  var:
     pedido: array [1..20] of integer;
     donde: array [1..20] of integer;
     i,tope: integer;
     ok: boolean;

      while(true) {
            pedido = obtener_pedido();
            for(i = 1; i <= 20; i++) {
                  if (pedido[i] == null) {
                        donde[i] = null;
                  }
                  else {
                        donde[i] = donde_esta(pedido[i]);
                  }
            }
            tope = i;
            Estanterias.obtener(donde, id);
            ok = true;
            for(i = 1; i < tope && ok; i++) {
                ok = hay_stock(pedido[i],donde[i]))
            }

            if(ok) {
                  for(i = 1; i < tope && ok; i++) {
                        retirar(pedido[i], donde[i]);
                  }
            }
            Estanterias.liberar(id);
            finalizar_pedido(ok);
      }
```

```
}

inspector()
{
  while(true)
  {
    int donde = donde_inspeccionar();
    Estanterias.inspeccionar(donde);
    inspeccionar(donde);
    Estanterias.fin_inspeccionar(donde);
  }
}
```