Examen Febrero de 2004

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.

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

Para las siguientes partes, conteste justificando adecuadamente, cada una de las preguntas.

1. ¿Qué permite la memoria virtual?

==========

Es una técnica que permite que la ejecución de un proceso aunque este no este completamente cargado en memoria. Esto hace que los procesos puedan ser más grandes que la memoria física (o real) del equipo.

==========

2. Enumere los pasos que sigue la rutina de fallo de página (page-fault)

==========

1. Encontrar en que lugar del disco esta la página que falló.
2. Encontrar un marco (página) libre en memoria principal:
   A. Si existe un marco libre, entonces pasar al punto 3.
   B. Si no existe un marco libre, entonces usar el algoritmo de remplazo de página del sistema para seleccionar a la víctima.
   C. Escribir en disco la página victima y actualizar la tabla de paginación.
3. Leer la página de fallo desde el disco y ponerla en el marco obtenido en el punto 2. Actualizar la tabla de paginación.
4. Volver al proceso del usuario.

==========

3.

a) Describa los algoritmos de reemplazo:

i. FIFO

ii. Óptimo

iii. LRU

==========

FIFO - Se reemplaza la página que hace más tiempo esta cargada en memoria principal.

Óptimo - Reemplazar la página que no será usada por el mayor período de tiempo. Para este algoritmo es necesario tener información de los pedidos de página futuros.

LRU - Reemplazar la página que hace más tiempo que no es accedida. (Least recently used)

==========

b) Para cada uno de ellos muestre la secuencia de reemplazo para los siguientes pedidos de página:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Asumiendo que el tamaño de la memoria principal es de 3 páginas.

==========

En todos los casos las primeras tres páginas entran a la memoria sin reemplazos. Los reemplazos se muestran en notación (página original, reemplazo).

FIFO: (7,2) (0,3) (1,0) (2,4) (3,2) (0,3) (4,0) (2,1) (3,2) (0,7) (1,0) (2,1)

Óptimo: (7,2) (1,3) (0,4) (4,0) (3,1) (2,7)

LRU: (7,2) (1,3) (2,4) (3,2) (0,3) (4,0) (0,1) (3,0) (2,7)

==========

4. Describa lo que es una dirección virtual (logical address) y una dirección real (physical address)

==========

Dirección virtual (lógica): Dirección generada por los programas a nivel de CPU.

Dirección real (física): Es una dirección sobre la memoria real (física).

==========

5. ¿En qué consiste el modelo de paginación simple (de un nivel)? Muestre cómo las direcciones virtuales se mapean a direcciones reales. Mencione todos los elementos involucrados.

==========

Paginación simple: La memoria real es dividida en bloques (frames) de tamaño fijo. La memoria virtual es también divida en bloques del mismo tamaño que son llamados páginas.

El tamaño de página esta dado por el hardware.

La traducción de una dirección virtual a real se hace así:

Las direcciones virtuales se dividen en 2 partes: un número de página y un offset:

```
 +-----------+------------+
 |    p      |     d      |
 +-----------+------------+
 Dirección virtual
```

p es un índice de la tabla de paginación (que en general la proveé el hardware).

Con p se va a la tabla y se obtiene f (frame de la memoria real), que junto con d forman la dirección real:

```
 +-----------+------------+
 |    f      |     d      |
 +-----------+------------+
 Dirección real
```

==========

6. Describa el modelo de segmentación. Muestre como las direcciones virtuales se mapean a direcciones reales. Mencione todas las estructuras y elementos involucrados.

==========

Segmentación es un esquema que soporta la perspectiva de la vista del usuario de lo que es la memoria.

La memoria se divide en segmentos de diferentes tamaños. Los segmentos están dados por la memoria usada por los programas.

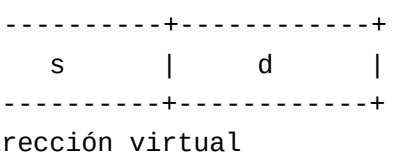
El espacio de las direcciones lógicas de un proceso es un conjunto de los segmentos.

Un ejemplo de un proceso:

Se tiene un segmento para las variables globales, otro para el stack, otro para el código del programa, otro para cada variable local de los procedimientos y funciones.

La traducción de una dirección virtual a real se hace así:

Las direcciones virtuales se dividen en 2 partes: un número de segmento y un offset:

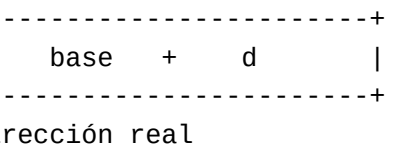


s es un índice sobre la tabla de segmentación (que en general la provee el hardware).

Cada entrada de la tabla de segmentación tiene 2 campos: límite y base.

El límite es el tamaño del segmento y la base es la dirección real de donde comienza el segmento en memoria real.

Por un lado se verifica que: d < límite, sino es así entonces se produce un trap de error. Por otro lado se hace la suma: base + d para obtener la dirección real



7.

a) Describa tres de los algoritmos vistos en el curso de planificación de corto plazo.

==========

Se presentan 4 algoritmos :

1.First-Come, First-Served

Primero en llegar a la ready-queue, primero en asignarle la CPU.

2.Shortest Job First

Dada una lista de procesos se le asigna la CPU al trabajo que le reste menor tiempo de ejecución.

Para la ejecución de este algoritmo es necesario que reporten el tiempo que le resta en alguna unidad común a todos.

3.Priority

A los procesos se les asigna una prioridad (p.ej: un numero natural). Se define una relación de orden de estas prioridades y según ella los procesos que tengan mayor prioridad tendrán acceso al CPU antes.

4.Round Robin

Se define un quantum de tiempo global a todos los procesos. Se asignan los procesos a la CPU y se les quita cuando termine el tiempo (quantum).

==========

b) Muestre a través de un diagrama de Gantt como sería la ejecución para el siguiente ejemplo:

Proceso	Tiempo de ejecución
P1	10
P2	5
P3	2
P4	4

Defina (especificando) a gusto datos adicionales que su algoritmo requiera (time slice, prioridad, etc.)

==========

1.First-Come, First-Served
Suponiendo un orden de llegada: P1, P2, P3 y P4

```
    P1        P2  P3  P4
|----------|-----|--|----|
```

2.Shortest Job First

```
 P3   P4   P2     P1
|--|----|-----|----------|
```

3.Priority
Asumiendo las siguientes prioridades:
P1 -> 4
P2 -> 3
P3 -> 2
P2 -> 1
Donde a mayor número mayor prioridad.

```
    P1        P2  P3  P4
|----------|-----|--|----|
```

3.Round-Robin
Asumiendo un quantum de tiempo igual a 4 y tomando los procesos en el orden P1, P2, P3 y P4

```
  P1   P2  P3  P4   P1 P2 P1
|----|----|--|----|----|-|--|
```

==========

Problema 2

El InCo desea involucrarse en la construcción de su nuevo reproductor de MP3. Para este se necesitan múltiples componentes, entre los que se destaca el medio de almacenamiento de los archivos musicales.

Para construir el almacenamiento de la música, se pensó en un disco duro típico de un computador. Lo que debe determinarse ahora, es el sistema de archivos que se pondrá en este. Para esto, se tienen los siguientes requerimientos.

- Cada canción esta almacenada en un archivo de extensión mp3. Cada archivo debe tener los siguientes atributos: nombre, fecha, duración (en minutos) y cantidad de bits de codificación (64, 128 o 256 Kbits por segundo). Una codificación de X Kbits por segundo indica que N segundos necesitan XN/8 Kbytes de espacio en disco.
- El sistema deberá almacenar múltiples álbumes. Un álbum contiene un conjunto de canciones o archivos mp3, existiendo un tope de 12 canciones por álbum. El álbum posee los siguientes atributos: nombre, fecha, artista, cantidad de canciones y genero.
- No existe un limite (mas allá del físico) de álbumes en el almacenamiento.
- Cada canción es accedida en forma secuencial cada vez que desea escucharse.
- El disco duro utiliza sectores de 8Kb bytes.
- Deben poder realizarse inserciones y borrados de canciones de cada álbum.
- Deben poder realizarse dos tipos de búsqueda:

  Por álbum:, en las cual dado un nombre de album los datos de la primer canción del album son devueltos.

  Por nombre de canción y artista: en la cual dados un nombre de canción y un nombre de artista, los datos de la primer canción que coincida son devueltos.

Se pide:

a) Definir las estructuras necesarias para implementar el sistema de archivos del reproductor MP3, cumpliendo con los requisitos planteados.

b) Indique como se almacena la información en estas estructuras en el almacenamiento, a fin de no perderlas cada vez que se apaga el reproductor, así como también las estructuras que son necesarias de mantener en memoria para el correcto funcionamiento del reproductor.

c) Implemente las funciones:

   `reproducir_album(string album)`, la cual recibe el nombre de un album y reproduce las canciones en él presentes.

   `reproducir_intro(string album)`, la cual recibe el nombre de un album y reproduce los primeros 10 segundos (o su mejor aproximación) de cada canción del album.

Nota: Se dispone de las siguientes funciones auxiliares

- `reproducir(byte[] datos)`: La cual inicia la reproducción de una canción.
- `store(byte[] datos, int sector)`: La cual almacena un conjunto de bytes en un sector del disco.
- `byte[] read(int sector)`: La cual recupera un conjunto de bytes de un sector del disco.

Solución:

a) Debemos indicar las distintas estructuras que componen nuestro sistema de archivos. Debido a que las canciones se almacenan en álbumes, y estos contienen los archivos con extensión mp3, utilizaremos una estructura jerárquica de tres niveles para representar esto.

En el nivel inicial tendremos el directorio raíz, del cual colgaran una serie de directorios, uno por cada álbum existente. Dentro de cada directorio se almacenara la información de cada archivo para cada canción del álbum

La cantidad máxima de álbumes estará dada por la arquitectura del disco. Cada álbum estará representado por un array de 12 entradas de directorio, donde cada entrada tendrá la forma siguiente:

```
T_entrada ::= Record
      Nombre: string;
      Fecha: date;
      Duracion: integer;
      Codificacion: integer;
      SectorInicial: integer;
      EnUso: boolean;
End;
```

Se agregan los atributos SectorInicial para indicar el sector de disco donde comienza la canción, y una marca booleana para indicar si la entrada en el álbum esta en uso. Si no esta en uso, no tienen validez los demás campos.

El álbum esta representado por la siguiente estructura:

```
T_album ::= Record
      Nombre: string;
      Fecha: date;
      Artista: string;
      Genero: string;
      TotalCanciones: integer;
      SectorInicial: integer;
      Canciones: array[1..12] of T_entrada;
End;
```

También aquí tenemos el sector inicial, indicando donde se encuentra la información del álbum en el disco.

Por ultimo, el directorio raíz esta representado por la siguiente estructura

```
T_raiz ::= lista(T_album)
```

Donde el directorio raíz contiene una lista de álbumes, accesibles con las operaciones tradicionales de una lista.

Debido a que el acceso a las canciones es secuencial, se utilizara una estructura tipo FAT, en donde un archivo estará compuesto por un conjunto sectores, enlazados en forma de lista. La fat es una estructura de la forma:

FAT ::= array[1..MAX_SECTORES_EN_DISCO] of integer;

Cada elemento de la fat contiene un numero de sector, indicado cual es el próximo sector que compone este archivo. Si tenemos el valor -1, entonces el archivo finaliza en dicho sector.

b) Las estructuras que componen el sistema de archivos, esto es, la FAT y el conjunto de álbumes con sus canciones, deberán ser almacenados en algún lugar del disco. Para esto debemos reservar una porción del mismo donde sabremos que se encuentra la FAT y el directorio raíz. De esta forma, con estos dos componentes podremos levantar desde el disco cualquier canción y álbum del sistema de archivos. Por lo general, se reserva el sector 0 del disco para almacenar la FAT y el directorio raíz, los cuales deberán mantenerse en memoria mientras el player este en funcionamiento.

c) Implementamos las funciones pedidas, asumiendo que la fat y el directorio raíz fueron cargados y se encuentran presentes en las variables globales:

```
FAT laFat;
lista(T_album) raiz;

function reproducir_album(string nombre_album) {
  // buscamos el album
  T_album album = null;
  encontrado = false;
  for (int i = 0; i < largo(raiz) && not encontrado; i++) {
      album = obtener_elemento(raiz, i);
      if (album.nombre == nombre_album) {
                  encontrado = true;
      }
  }

  // Si lo encontramos, iteramos por las canciones reproduciéndolas
  if (encontrado) {
      for (int i = 1; i < 12; i++) {
            if (album.canciones[i].EnUso) {
                  // Traemos del disco los datos de la cancion.
                  // Debemos iterar por la FAT hasta tener todos
                  // los sectores
                  int SecIni = album.canciones[i].SectorInicial
                  byte[] datos = cargarDesdeFAT(SecIni);
                              reproducir(datos);
            }
```

```
        }
      }
  }

  function reproducir_intro(string nombre_album) {
    // buscamos el album
    T_album album = null;
    encontrado = false;
      for (int i = 0; i < largo(raiz) && not encontrado; i++) {
            album = obtener_elemento(raiz, i);
            if (album.nombre == nombre_album) {
                  encontrado = true;
            }
      }

      // Si lo encontramos, iteramos por las canciones reproduciendo
      // solo los 10 primeros segundos de las mismas.
      if (encontrado) {
            for (int i = 1; i < 12; i++) {
                  if (album.canciones[i].EnUso) {
                        // Traemos del disco los datos de la cancion.
                        // Debemos iterar por la FAT hasta tener todos
                        // los sectores
                        int SecIni = album.canciones[i].SectorInicial
                        byte[] datos = cargarDesdeFAT(SecIni);

                        int cod = album.canciones[i].Codificacion
                        int cntBytes = (cod * 10 / 8) * 1024
                        byte[] datos = tomarBytes(datos, cntBytes);
                        reproducir(datos);
                  }
            }
      }
  }
================================================================================
```

Problema 3

Sea un array de 100 elementos accedido por un número no acotado de tareas concurrentes, donde cada tarea desea modificar un único rango de elementos consecutivos en un intervalo [x,y].

Se desea implementar la mutuaexclusión de dichos rangos de elementos utilizando semáforos.

Cada proceso debe bloquear únicamente los elementos de su rango, sin bloquear regiones innecesarias. Por ejemplo, no deben bloquearse los 100 elementos del array si se quiere modificar un rango que incluye 5 elementos.

Se pide:

a)

i. Resolverlo sin utilizar tareas auxiliares y con menos de 150 semáforos. Las tareas solicitan / liberan el rango de la siguiente manera:

```
solicito_rango( X, Y );
      :
//Operaciones sobre el rango
      :
libero_rango( X, Y );
```

ii. Analice el comportamiento de su solución de la parte (a.i) si se permitiera a las tareas solicitar más de un rango, de esta manera:

```
solicito_rango( X, Y );
solicito_rango( X1, Y1 );
...
solicito_rango( Xn, Yn );
      :
//Operaciones sobre los rangos
      :
libero_rangos( {X,Y}, {X1,Y1}, ... ,{Xn,Yn} );
```

==========

Solución:

```
TipoElem[100] elementos;
 Semaphore [100] locks;
 // Inicialización de los locks
 for (int i = 0; i < 100; i++) {
       init(locks[i], 1);
 }

 procedure solicito_rango (int desde, int hasta) {
       for (int i = desde; i < hasta; i++) {
             P(locks[i]);
       }
 }
```

```
 procedure libero_rango(int desde, int hasta) {
       for (int i = desde; i < hasta; i++) {
            V(locks[i]);
       }
 }
```

==========

b) Resolver el problema con las siguientes restricciones:
- el array es accedido por 10 tareas concurrentes.
- no se pueden utilizar mas de 25 semáforos.
- se permite utilizar tareas auxiliares.

==========

Solución:

En esta solución vamos a utilizar una tarea extra (Administrador) que será encargada de administrar los recursos.
Se desea obtener la mayor concurrencia posible, por lo cual solo tomaremos recursos cuando sea posible obtener todo el rango.

```
Const MAX_TAREAS = 10
Const LIBRE = -1;
Enum TipoPedido = [PEDIR, LIBERAR];
Semaphore memoria;
Semaphore admin;
Semaphore [MAX_TAREAS] mutexTarea;
TipoElem[100]      elementos;
Bits [100] mapaBits;


// La estructura Shared se utilizara para comunicar las tareas
// con el Administrador
Struct Shared
{
       int desde;
       int hasta;
       int tareaId;
       TipoPedido tipoPedido
};

// Esta estructura es utilizada por el Administrador para saber los
// pedidos pendientes
Struct TipoRango
{
       int desde;
        int hasta;
};
```

```
void inicializar()
{
      // Inicialización de estructuras de datos

      // Inicialmente todos los recursos están libres
      for (int i = 0; i < 100; i++)
            Bits[i] = 0

      //Inicialización de los semáforos
      for (int i = 0; i < MAX_TAREAS; i++)
            init(mutexTarea[i], 0);
      init (admin, 0);
      init (memoria 1);
}

// Esta función chequea el rango y retorna true si esta completamente disponible
function chequeoRango(int desde, int hasta):boolean
{
       for(int i = desde; i < hasta; i++)
       {
            if(mapaBits[i] == 1)
                  return false;
       }
       return true;
}

// Este procedimiento corre en la tarea Administrador
procedure admin_pedido()
{
       TipoRango [MAX_TAREAS] rango;
      int tareaId, desde, hasta;
       TipoPedido tipoPedido;

       // Inicialmente no tengo ningun pedido pendiente
       for (int i = 0; i < 10; i++)
            rango[i].desde = LIBRE;

      while(true)
       {
            P(admin);  // Espero alguna solicitud
            // Obtengo los datos de la solicitur
            tareaId = Shared.tareaId;
            desde = Shared.desde;
            hasta = Shared.hasta;
            tipoPedido = Shared.tipoPedido;
            V(memoria);  // Libero el mutex de los parámetros de la solicitud
            if(tipoPedido = PEDIR)
```

```
            {
                  // Verifico si tengo disponible todo el rango
                  if(chequeoRango(desde, hasta))
                  {  // En ese caso obtengo todos los elementos
                        for(int i = desde; i < hasta; i++)
                              mapaBits[i] = 1;
                       V(mutexTarea[tareaId]); //notifico a la tarea que
                                               //la solicitud fue satisfecha
                  }
                  else
                 {  // Si no puedo obtener el rango, dejo pendiente la solicitud
                        rango[tareaId].desde = desde;
                        rango[tareaId].hasta = hasta;
                  }
             }
            else
            {
                  // Libero el rango
                  for(int i = desde;  i < hasta; i++)
                        mapaBits[i] = 0;

                  // Ahora veo si puedo otorgar solicitudes pendientes
                  for(int i = 0; i < MAX_TAREAS; i++)
                  {
                        if(rango[i].desde != LIBRE)
                        { // Si tengo una solicitud pendiente para la tarea i
                              if(chequeoRango(rango[i].desde, rango[i].hasta)
                              {  // Si puedo otorgar la solicitud
                                  rango[i].desde = LIBRE;//Libero la solicitud
                                   for(int j = rango[i].desde; i<rango[i].hasta;
                                                                          j++)
                                          mapaBits[j] = 1;
                                    // Notifico a la tarea i que su solicitud
                                    // fue satisfecha
                                     V(mutexTarea[i]);
                              }
                        }
                  }
             }
       }
}

procedure solicito_rango(int desde, int hasta)
{
       P(memoria); // Obtengo el mutex de los parámetros de la solicitud
       Shared.tareaId = dameId(); // dameId() retorna el numero de
                                  // tarea (0..MAX_TAREAS)
       Shared.desde = desde;
```

```
        Shared.hasta = hasta;
        Shared.tipoPedido = PEDIR;
        V(admin);  // Despierto al Administrador para que atienda mi solicitud
        P(mutexTarea[taredId]); //Espero a que el Administrador me notifique que
                                //la solicitud fue satisfecha
}


procedure libero_rango(int desde, int hasta)
{
        P(memoria); // Obtengo el mutex de los parámetros de la solicitud
        Shared.tareaId = dameId();  // dameId() retorna el numero de
                                    // tarea (0..MAX_TAREAS)
        Shared.desde = desde;
        Shared.hasta = hasta;
        Shared.tipoPedido = LIBERAR;
        V(admin);  // Despierto al Administrador para que atienda mi solicitud
        P(mutexTarea[taredId]); // Espero a que el Administrador me notifique que
                                // la solicitud fue satisfecha
}
```

==========