Solución - Examen Julio de 2005

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

1. **¿Cuál es la principal ventaja de la multiprogramación?**

   Logra el incremento de utilización de CPU. Esto se consigue al cargar varios procesos en memoria y alternar la ejecución cuando se producen requerimientos de E/S por ejemplo.

2. **¿De qué forma se puede garantizar que todos los accesos a los dispositivos, por parte de un proceso, se deban hacer a través del sistema operativo?**

   Obligando a que las instrucciones de hardware que acceden al dispositivo sean privilegiadas. De estar forma, solo el sistema operativo puede ejecutarlas.

3. **Describa tres servicios que debe brindar el sistema operativo.**

   Ejecución de programas: realiza todas las tareas relativas a la ejecución de un programa (carga, linkeo dinámico, fin de ejecución, etc) en nombre del usuario
   Operaciones de I/O: brinda una interfaz abstracta sobre los dispositivos de I/O, permitiendo al usuario ignorar los detalles de bajo nivel de acceso a los mismos. Esto permite además lograr una mayor eficiencia y un manejo más seguro de los mismos
   Manipulación del filesystem: Permite al usuario realizar operaciones de alto nivel (lectura, escritura, etc) sobre archivos almacenados en diferentes medios. Esto permite al usuario lograr un almacenamiento persistente de sus datos y programas.
   Comunicaciones: brinda mecanismos de intercambio de información entre procesos, que se pueden encontrar en la misma máquina o no. Dichos mecanismos se pueden implementar mediante memoria compartida o pasaje de mensajes.
   Detección de errores: brinda mecanismos para aislar a los usuarios de posibles errores de hardware, o errores producidos por procesos de otros usuarios.
   Además, brindan asignación de recursos, contabilización y protección en sistemas multiusuario.

4. **¿Qué son los llamados al sistema (system calls)?**

   Los llamados al sistema proveen una interfaz entre los procesos y el sistema operativo a través de la cual los procesos pueden acceder a los servicios que este brinda.

5. **Para hacer un llamado al sistema, ¿es necesario estar en modo monitor o de usuario? Justifique.**

   Los llamados al sistema son ejecutados por procesos de usuario y por lo tanto ejecutan en modo usuario. Para procesar el llamado el sistema operativo trabaja en modo monitor y una vez finalizado, antes de retornar el control al proceso, vuelve a modo usuario.

6. **En el diseño de un sistema operativo, ¿cuál debiera haber ido la justificación de elegir un sistema monolítico frente a un diseño por capas?**

   La necesidad de implementar un sistema con mayor rendimiento. El sistema zoolítico es más eficiente ya que disminuye la carga de la comunicación entre capas.

7. **En una arquitectura multiprocesador, ¿sería conveniente utilizar un sistema operativo cuyas primitivas o servicios no sean en su mayoría reentrantes?. Justifique.**

   No, dado que el código no reentrante si es ejecutado simultáneamente por más de un proceso puede generar inconsistencias.

8. **En una arquitectura multiprocesador, describa las ventajas y desventajas de contar con cache a nivel de los procesadores.**

   La cache a nivel de procesador incrementa el desempeño ya que los procesadores pueden obtener los datos de la cache (copias de los datos en memoria principal) y evitar la búsqueda en la memoria principal. Por otro lado, se debe mantener la coherencia entre los datos de las caches lo cual vuelve más complejo el hardware necesario.

9. **En una arquitectuela multiprocesador con segmentación, describa los pasos y dificultades en ocasión de cambiar un segmento de lugar como resultado de reorganizar la memoria.**

   Localizar espacio libre contiguo de tamaño suficiente para ubicar el segmento a mover. Bloquear los procesos que referencial al segmento físico a reubicar. Mover el segmento. Modificar las tablas de segmento de los procesos antes mencionados (colocando en la entrada correspondiente el nuevo valor base y límite). Desbloquear los procesos.

10. **Mencione las características principales de la protección y el compartimiento de segmentos en una arquitectura con manejo de memoria a través de segmentación.**

    Dado que existen segmentos de instrucciones de código y de datos, a cada uno de ellos se les puede asociar un modo de acceso (lectura/escritura). De esta forma, se pueden generar controles a través de hardware que permitan garantizar el acceso a los mismos por parte de los programas

    Compartir segmentos de códigos (que son segmentos de lectura), permite maximizar el uso de la memoria principal. Los procesos que comparten un segmento de código, solo manejan la referencia al segmento compartido por ambos, y no hay necesidad de tener varias copias de un mismo segmento.

11. **Explique como es un sistema RAID 0+1 y describa sus ventajas.**

    El nivel de RAID 0+1 fracciona los datos ("stripping") para mejorar el rendimiento, pero también utiliza un conjunto de discos duplicados para conseguir redundancia de datos de manera de aumentar la tolerancia ante fallas. Al ser una variedad de RAID híbrida, RAID 0+1 combina las ventajas de rendimiento de RAID 0 con la redundancia que aporta RAID 1.

12. **En el diseño de un sistema de archivos tipo Unix, ¿ud. Pondría el nombre de archivo dentro del i-nodo? Justifique.**

    No, en unix los inodos contienen un número único que identifica el archivo. El nombre se maneja en la estructura del sistema de archivos de manera de poder referenciar al mismo archivo con distinto nombre.

13. **Si se dispone de un sistema operativo que no implementa el concepto de memoria virtual, ¿qué problemas identifica en la ejecución de los procesos? ¿Podría existir la multiprogramación en ese sistema?**

    No podrían existir procesos de tamaño mayor que la memoria física disponible ya que los procesos deben cargarse completamente en memoria para ejecutar. Si puede existir multiprogramación, limitada por la cantidad de procesos que puedan cargarse completos simultáneamente.

Problema 2

Se dispone de un sistema operativo que almacena sus datos en un sistema de archivos tipo XSFS (eXtended Secure File System). Este sistema tiene algunas modificaciones con respecto a los clásicos File Sytems Indizados (tipo Unix), ya que:

- Permite almacenar los datos en múltiples discos duros. Esto es, un archivo puede estar disperso a lo largo de múltiples dispositivos.
- Los archivos del mismo tienen asociados un conjunto de usuarios los cuales podrán operar con los mismos (altas, bajas, lecturas y escrituras).
- Los discos poseen sectores de 1024 bytes. Si un archivo utiliza parte de un sector, entonces ese sector es inaccesible para otro archivo.

Este filesystem tiene las siguientes restricciones:

Es plano, es decir que solamente se dispone del directorio raíz. Puede asumir también que el directorio raíz se encuentra en una posición fija definida por usted.

Usa un esquema indizado simple, donde cada i-nodo referenciará directamente N bloques de datos.

Se dispone de las siguientes funciones y procedimientos auxiliares:

- **busca_libre(in tabla): integer**

  Dada una tabla (según su definición) retorna un índice en la misma donde exista una entrada libre.

- **read (disco, sector) : bytes [], write (disco, sector, datos)**

  La función *read* recibe la dirección física donde se encuentran los datos a leer y retorna un arreglo de bytes con los datos allí presentes.

  Análogamente, la función *write* recibe la dirección física y los datos a guardar.

Se pide:

a) Definir las estructuras de datos del XSFS, de manera tal que los requerimientos.

anteriores puedan ser cumplidos. Explique la necesidad y función de cada estructura.

b) Implementar el procedimiento **xsfs_copy** que copia un archivo. Este tiene el siguiente prototipo:

xsfs_copy(in origen: string, in destino: string)

origen: Nombre del archivo origen.

destino: Nombre del archivo destino.

Notas:

Se puede asumir que el volumen ya está montado al iniciar esta operación.

No es necesario manejar operaciones a nivel de bits en enteros, puede definir estructuras con miembros de tipo boolean.

El archivo destino debe quedar con los mismos permisos que el archivo origen.

Debe verificar que el usuario que ejecuta xsfs_copy tenga permisos sobre el archivo origen. Para este fin se dispone de la primitiva **get_user ()** que nos retorna el nombre del usuario que la ejecuta.

a)

```
type vector_bits: array [1..MAX_DISCS][1..MAX_SECTORS] of boolean;    // indica los sectores libres

type tabla_inodos: array[1..CANT_INODOS] of inodo

type inodo:
      record of
            usado: boolean
            tam_bits: integer  // tamaño en bits
            tam_bloques: integer // tamaño en bloques
            datos: array[1..N] of direccion
      end

type direccion:
      record of
            disco: integer
            sector: integer
      end

type directorio:
      record of
            nombre: string
            inodo: integer
            ocupado: bolean
            usuarios: set of string
      end
```

b)

```
var tab: tabla_inodos;
```

```
      var vector: vector_bits;

      procedure xsfs_copy (in origen: string, in destino: string)
      var   datos: byte[]
            dir: directorio[]
            i, j, k, inodo: integer
            fin: boolean

      begin
            datos = read (raiz.disco, raiz.sector);
            dir = (directorio)datos;
            i := 0;
            fin := false;
            while not fin and i <= MAX_DIR do
            begin
                  if dir[i].ocupado and dir[i].nombre = origen and
                                          getUser() in dir[i].usuarios then
                        fin := true;
                  else
                        i := i+1;
            end;
            if fin then
            begin
                  j := buscar_libre(dir)
                  if j >= 0 then
                  begin
                        dir[j].ocupado := true;
                        dir[j].nombre := destino;
                        dir[j].usuarios := dir[i].usuarios;
                        inodo := buscar_libre(tab);
                        dir[j].inodo = inodo;
                        for k := 1 to tab[dir[i].inodo].tam_bloques do
                        begin
                              tab[inodos].datos[k] :=
                                                      buscar_libre(vector);
                              marcar_ocupado(vector,
                                                    tab[inodos].datos[k]);
                              datos := read
(tab[dir[i].inodo].datos[k].disco,  tab[dir[i].inodo].datos[k].sector);
                              write(datos, tab[inodos].datos[k].disco,
```

```
                                        tab[inodos].datos[k].sector);
                    end
                    tab[inodos].tam_bloques :=
                                        tab[dir[i].inodo].tam_bloques;
                    tab[inodos].tam_bits :=
                                           tab[dir[i].inodo].tam_bits;
                    tab[inodos].usado := true;
               end
          end
     end
```

Problema 3

Se desea modelar utilizando mailboxes el siguiente problema:

El taller mecánico "*Boxes*" tiene capacidad para 20 vehículos en sus instalaciones. La entrada de los usuarios a las instalaciones debe ser estrictamente en orden de llegada (la solución no debe permitir bajo ningún concepto "colados"). El usuario al entrar le dará su número de socio al **Portero** el cual indicará si tiene la cuota al día o si debe pagar la cuota antes de pedir ser atendido por los mecánicos.

El usuario debe ser atendido por el primer mecánico libre de los 5 con que cuenta el taller. El mecánico una vez terminado el arreglo le indicará a la caja el número de socio y el monto a cobrar por el arreglo.

El cliente recibirá de la caja el monto a pagar.

Se dispone de las siguientes funciones:

- **mi_numero():integer**
  - Invocada por el usuario le indica su número de socio.
- **al_dia(Nro_socio):boolean**
  - Invocada por el portero para ver si el socio tiene la última cuota paga.
- **pago_cuota(Nro_socio)**
  - Invocada por el usuario para pagar las cuotas pendientes.
- **arreglar_auto(Nro_socio):integer**
  - Invocada por un mecánico para arreglar el auto. Retorna el costo del arreglo.
- **pagar_arreglo(Nro_socio, Monto)**
  - Invocada por el usuario para pagar el monto que le fue indicado por la caja.

Notas:

- Se prohibe expresamente el busy-waiting.
- Se debe explicitar la semántica de las primitivas de mailbox utilizadas. La no especificación completa es causal de pérdida del ejercicio.

```
Mailboxes infinitos
Send no bloqueante, receive bloqueante

taller_mutex: mailbox (nil);
portero_in: mailbox (integer, string)
portero_out: mailbox (integer)
portero_mutex: mailbox (nil)
espera: array [1..20] of mailbox (nil)
mecanico[1..5]: mailbox (integer, integer);
```

```
caja_in: mailbox (integer, integer);

Procedure cliente ()
Var   alDia, lugar, libre, monto: integer;
Begin
      receive(taller_mutex);
      receive(portero_mutex);
      send(portero_in, (mi_numero(), CLI_ENTRA));
      (alDia, lugar) = receive(portero_out);
      send(portero_mutex, nil);
      if not alDia then
            ponerse_al_dia(mi_numero());
      libre = receive(espera[lugar]);
      send(mecanico[libre], (mi_numero(), lugar));
      monto = receive(espera[lugar]);
      pagar_arreglo(mi_numero(), monto);
      send(portero_in, mi_numero(), CLI_SALE);
      send(taller_mutex);
End


Procedure portero()
Var   socio, libre: integer;
      alDia: boolean;
      esperando: array [1..20] of integer;
      cola_espera: cola of integer;
      mec_libre: array[1..5] of boolean;
Begin
      for i:= 1 to 5
            mec_libre[i] = true;
      for i:= 1 to 20
            esperando[i] = 0; // libre

      while true do
      begin
            (socio, tipo) = receive(portero_in);
            if tipo = 'CLI_ENTRA' then
            begin
                  alDia = al_dia(socio);
                  lugar = darLibre(esperando);
```

```
                  esperando[lugar] = socio;
                  send(portero_out, (alDia, lugar));
                  if hayLibre (mec_libre) then
                  begin
                        libre = darLibre(mec_libre)
                        send(espera[lugar], libre);
                        mec_libre[libre] = false;
                  end
                  else
                        cola_espera.push(lugar);
            end;
            else if tipo = 'CLI_SALE' then
            begin
                  lugar = buscar(socio, esperando);
                  esperando[lugar] = 0;
            end;
            else if tipo = 'MEC_LIBRE' // aca socio es el num de mecanico
            begin
                  if empty(cola_espera) then
                        mec_libre[socio] = true;
                  else
                        send(espera[cola_espera.pop()], socio);
            end
      end
End


Procedure mecanico (num: integer)
Var socio, monto, lugar: integer;
Begin
      while true do
      begin
            receive(mecanico[num], (socio, lugar));
            monto = arreglar_auto(socio);
            send(caja_in, (monto, lugar));
            send(portero_in, (num, 'MEC_LIBRE'));
      end
End
```

```
Procedure caja ()
Var monto, lugar: integer;
Begin
      while true do
      begin
            (monto, lugar) = receive (caja_in);
            send (espera[lugar], monto);
      end
End

Begin
      send(portero_mutex, nil);
      for i:= 1 to 20
            send(taller_mutex, nil);
      cobegin
            for i:= 1 to 5
                  mecanico(i);
            portero();
            caja();
            clientes...;
      coend
End
```