Examen Marzo 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.

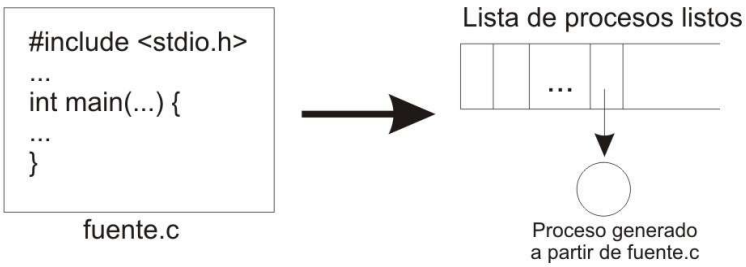
Finalización

El examen dura 4 horas.

Problema 1

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

1. Describa lo que entiende por sistema altamente acoplado/integrado.
2. ¿ Qué es lo que implementan los manejadores de los dispositivos (device drivers) ?
   ¿ Forman parte del sistema operativo ?. Justifique.
3. Enuncie y describa brevemente 3 tipos de llamados al sistema (system calls).
4. ¿ Cuales son las etapas previas por las que debió pasar un proceso que se encuentra en la lista de procesos listos, a partir de un archivo que contiene un código fuente escrito en lenguaje C ?



5. Enuncie y describa una herramienta que brindan los sistema operativos de tipo UNIX para permitir la sincronización entre los procesos.
6. Escriba un algoritmo en lenguaje de alto nivel (C o PASCAL) que solucione el problema de la sección crítica para dos procesos en un sistema monoprocesador. Asuma que las sentencias son ejecutadas de forma atómica. Solo se permitirá el uso de variables globales a los dos procesos, y es posible utilizar 'busy waiting'.
   ¿ Que requerimientos debe satisfacer su solución para que esta sea correcta ? Solo enuncielos, no los describa.
7. En un sistema monoprocesador: ¿ Són necesarias la operaciones 'Test and Set' y 'Swap' ? Justifique. Si considera que es necesario: ¿ Quién provee la operación 'Test and set' ?
8. ¿ El espacio de direcciones virtuales de un proceso depende de la arquitectura (hardware) o del sistema operativo ?
9. Según su respuesta anterior: Si tenemos una arquitectura de 64 bits con un sistema operativo en 32 bits. ¿ Cual es el tamaño máximo de direccionamiento posible para un proceso ?
10. Enumere dos ventajas de utilizar librerías dinámicas en vez de librerías estáticas.
11. Con la estructura Inodo utilizada en sistemas de archivos tipo UNIX ¿ Qué tipos de archivos se pueden representar ?
    a. Archivo (de texto, ejecutables, datos, etc).
    b. Directorios.
    c. Archivos y directorios.
    d. Archivos, directorios y enlaces débiles (soft link).
    e. Archivos, directorios, enlaces débiles y fuertes (hard link).

    Especifique la mejor opción y justifique brevemente.
12. Mencione los atributos más importantes de un inodo.
13. Sea un disco con 32 cilindros numerados del 0 al 31, y la siguiente cola de pedidos:

    12, 18, 1, 15, 2, 13, 17, 6, 22 y 16.

    Haga una tabla que muestre la secuencia de planificación de atención de los pedidos para los algoritmos First Come First Served, Shortest Seek Time First, SCAN, C-SCAN, LOOK y C-LOOK. Suponga que la cabeza del disco está ubicada sobre el cilindro 30.

    Nota: En los algoritmos que pueda haber más de una opción, elija una y aclare el criterio.

Problema 2

La heladería “La Marsellesa” tiene 4 vendedores, y vende cucuruchos de un solo gusto de entre 20 disponibles.

Los clientes que ingresan se forman en una única fila por orden de llegada, hasta que un vendedor pueda atender al primero, y así sucesivamente. La cantidad máxima de clientes no es conocida.

La heladería tiene un recipiente por cada gusto. Los vendedores no pueden extraer helado a la vez del mismo recipiente.

Se pide: Modelar en Ada las tareas Cliente y Vendedor.

Se dispone de los siguientes procedimientos:

- **que_helado_quiero(out:gusto)** Llamada por los clientes y que devuelve el gusto de helado que va a pedir.
- **armar_helado(in:gusto)** Llamada por los vendedores para poner el helado en el cucurucho.
- **entregar_o_recibir_helado()** Deberá ser ejecutado por el cliente o el vendedor para que el cliente reciba el helado (solamente debe ser llamada por una de las dos tareas).
- **comer_helado()** Llamada por los clientes para comer el helado que le sirvieron

Nota: Se pueden utilizar tareas auxiliares

Problema 3

Se dispone de un sistema de memoria virtual de segmentación (pura) de largo variable, el cual utiliza First Fit como técnica de ubicación de segmentos.

Se pide:

a) Describir brevemente el mecanismo de direccionamiento y el formato de dirección de memoria virtual. Definir la estructura de datos necesaria para representar el descriptor de un segmento.

b) Implementar la función **First_Fit(largo: long): long**

Esta función retorna la dirección de memoria donde existe un bloque libre de al menos tamaño **largo** o -1 si no se pudo encontrar dicho bloque.

c) Implementar el procedimiento de creación de segmentos

Instancio_Segmento(largo: long): descriptor

Para implementar esta función, se dispone de las siguientes primitivas:

- **Compactar (): long**
  Esta función compacta los bloques libres generando un único bloque cuyo tamaño es la suma de los bloques libres. Retorna el tamaño de dicho bloque.
- **Reemplazar (largo: long)**
  Baja a disco la mínima cantidad de segmentos cuyo tamaño sume al menos **largo**.

El sistema no debe:

- Compactar los bloques libres innecesariamente.
- Bajar a disco segmentos de manera innecesaria.

==========

Soluciones

==========

Problema 1.

1. Describa lo que entiende por sistema altamente acoplado/integrado.
   Es un sistema multiprocesador en donde los procesadores comparten un bus común de alta velocidad, el reloj, la memoria y los dispositivos.

2. ¿Qué es lo que implementan los manejadores de los dispositivos (device drivers)? ¿Forman parte del sistema operativo?. Justifique.
   Implementan primitivas definidas por el sistema operativo utilizando el lenguaje del dispositivo.
   Sí, el sistema operativo es el encargado de administrar el hardware, y de presentar al usuario una interfaz para el uso de los distintos componentes del sistema.

3. Enuncie y describa brevemente 3 tipos de llamados al sistema (system calls).
   a. Control de procesos.
   b. Manipulación de archivos.
   c. Manipulación de dispositivos.
   d. Información de mantenimiento.
   e. Comunicación.

4. ¿Cuales son las etapas previas por las que debió pasar un proceso que se encuentra en la lista de procesos listos, a partir de un archivo que contiene un código fuente escrito en lenguaje C?
   Se debe primero compilar (compile), luego ensamblar (link) con los demás códigos objetos para crear un archivo ejecutable. Finalmente, el cargador del sistema lo debe cargar (loader).

5. Enuncie y describa una herramienta que brindan los sistema operativos de tipo UNIX para permitir la sincronización entre los procesos.
   Semáforos. Son primitivas de sincronización. En su implementación se utilizan instrucciones de máquina que permiten la atomicidad de la operación.

6. Escriba un algoritmo en lenguaje de alto nivel (C o PASCAL) que solucione el problema de la sección crítica para dos procesos en un sistema monoprocesador. Asuma que las sentencias son ejecutadas de forma atómica. Solo se permitirá el uso de variables globales a los dos procesos, y es posible utilizar 'busy waiting'.
   ¿Que requerimientos debe satisfacer su solución para que esta sea correcta? Solo enuncielos, no los describa.

```
-----------------------------------
flag[i] := true;
turn := j;
while (flag[j] and turn=j) do no-op;

      SECCION CRITICA

flag[i] := false;
-----------------------------------
```

   a. Mutuo Exclusión.
   b. Progreso.
   c. Espera limitada.

7. En un sistema monoprocesador: ¿Són necesarias la operaciones 'Test and Set' y 'Swap'? Justifique. Si considera que es necesario: ¿Quién provee la operación 'Test and set'?
   Sí, son necesarias ya que la sincronización entre los procesos se debe proveer al igual que en un sistema multiprocesador. La instrucción es provista por el hardware.

8. ¿El espacio de direcciones virtuales de un proceso depende de la arquitectura (hardware) o del sistema operativo?
   De los dos. La arquitectura le impone una cota máxima, pero el software puede ejecutar en niveles menores.

9. Según su respuesta anterior: Si tenemos una arquitectura de 64 bits con un sistema operativo en 32 bits. ¿Cual es el tamaño máximo de direccionamiento posible para un proceso?
   32 bits.

10. Enumere dos ventajas de utilizar librerías dinámicas en vez de librerías estáticas.
    a. Se aprovecha mejor el espacio en disco.
    b. Es más fácil cambiar una versión de una biblioteca que mantenga los cabezales (headers) ya que no es necesario re-compilar los códigos fuentes de los programas.
    c. Mejor utilización de la memoria volátil (RAM). El sistema operativo carga una única copia de la biblioteca dinámica. Los procesos que la utilizan hacen referencia sin necesidad de cargarla para cada uno.

11. Con la estructura Inodo utilizada en sistemas de archivos tipo UNIX ¿ Qué tipos de archivos se pueden representar ?
    a. Archivo (de texto, ejecutables, datos, etc).
    b. Directorios.
    c. Archivos y directorios.
    d. Archivos, directorios y enlaces débiles (soft link).
    e. Archivos, directorios, enlaces débiles y fuertes (hard link).

    Especifique la mejor opción y justifique brevemente.

    d. Los enlaces fuertes son referencias a nivel de la estructura de nombrado del sistema de archivos y no a nivel de Inodo.

12. Mencione los atributos más importantes de un inodo.
    a. Punteros a los bloques de datos directos e indirectos.
    b. Tamaño del Inodo.

13. Sea un disco con 32 cilindros numerados del 0 al 31, y la siguiente cola de pedidos:

    12, 18, 1, 15, 2, 13, 17, 6, 22 y 16.

    Haga una tabla que muestre la secuencia de planificación de atención de los pedidos para los algoritmos First Come First Served, Shortest Seek Time First, SCAN, C-SCAN, LOOK y C-LOOK. Suponga que la cabeza del disco está ubicada sobre el cilindro 30.

    Nota: En los algoritmos que pueda haber más de una opción, elija una y aclare el criterio.

    En los casos SCAN, C-SCAN, LOOK y C-LOOK supondremos que la cabeza se viene moviendo de izquierda a derecha.

    FCFS: 30-12-18-1-15-2-13-17-6-22-16.

    SSTF: 30-22-18-17-16-15-13-12-6-2-1.

    SCAN: 30-31-22-18-17-16-15-13-12-6-2-1.

    CSCAN: 30-31-0-1-2-6-12-13-15-16-17-18-22.

LOOK: 30-22-18-17-16-15-13-12-6-2-1.
CLOOK: 30-0-1-2-6-12-13-15-16-17-18-22.

==========

Problema 2.

```
vend : array [1..10] of Vendedor;
adminGustos: array [1..20] of AdminGusto;


Task AdminCola is
      Entry nuevoCliente (numVend: out Integer);
      Entry finAtencion (numVend: in Integer);
End AdminCola

Task Body AdminCola is
      ocupados = array [1..10] of Boolean;

Begin
      for i:= 1 to 10 do
            ocupados[i] = false;
            vend[i].recibirNum (i);
      end;

      loop
            select
                  accept finAtencion (numVend: in Integer) do
                        ocupados[numVend] = false;
                  end
            or
                  when (hayLibre(ocupados)) =>
                  accept nuevoCliente (numVend: out Integer) do
                        numVend = darLibre(ocupados);
                        ocupados[numVend] = true;
                  end
            end
      end
End AdminCola


Task body Cliente is
      vendedor, gusto: Integer

Begin
      que_helado_quiero(gusto);
      AdminCola.nuevoCliente(vendedor);
      vend[vendedor].atender(gusto);
      comer_helado();
End Cliente


Task type Vendedor is
      Enty recibirNum (num: in Integer);
      Entry atender(gusto: in Integer);
End Vendedor
```

```
Task body Vendedor is
      miNum: Integer;

Begin
      accept recibirNum (num: in Integer)
            miNum = num;
      end;

      loop
            accept atender (gusto: in Integer)
                  adminGustos[gusto].Acceder;
                  armar_helado(gusto);
                  adminGustos[gusto].Liberar;
                  entregar_o_recibir_helado();
            end
            AdminCola.finAtencion(miNum);
end
End Vendedor


Task type AdminGusto is
      Entry Acceder;
      Entry Liberar;
End AdminGusto

Task body AdminGusto is
Begin
      loop
            accept Acceder;
            accept Liberar;
      end
End AdminGusto
```

==========

Problema 3.

a)
Las direcciones de memoria virtual son una pareja (nro. de segmento, desplazamiento). Para encontrar la imagen de un objeto en memoria real el operando de una instrucción será un descriptor de segmento (de aquellos del proceso en cuestión).

Un descriptor de segmento contiene:

- dirección del segmento (memoria real si esta residente, disco si no).
- bit de residencia.
- largo del segmento.
- otros (bits de protección por ejemplo).

El bit de residencia indica si el segmento está en memoria principal o no; si no está residente su referencia provocará un PAGE FAULT invocando al Sistema para que lo cargue desde memoria secundaria.
A la dirección real así obtenida se le suma el desplazamiento dado con lo que se tiene la dirección real buscada.

```
Descriptor_Segmento = Record
                        Direccion: Long
                        Residente: Boolean
```

```
                        Largo: Long
                      End Record
```

b)

Las técnicas de ubicación nos permiten decidir que parte de la memoria libre se usará para satisfacer un determinado pedido de recurso memoria (por ejemplo para alojar un segmento de un programa que se activa). Best Fit elige el segmento que mejor se aproxime al tamaño requerido. Tiene overhead ya que debe recorrer todos los segmentos a fin de comparar o mantener la lista ordenada. Como contrapartida aumenta el número de asignaciones exactas o casi con lo que los sobrantes serán de pequeño tamaño.

Implementaremos First Fit al instanciar un segmento.

```
type m_libre = record
            proxima : long;
            largo : integer;
      end;

var M : long;  {lista de memoria libre}

Function First_Fit (largo: long) : long

var dir, actual : long;
encontre : boolean;
begin
      actual := M;
      if actual <> NIL then     {hay algun segmento libre}
            if actual^.largo >= l then  {el primero sirve}
                  if actual^.largo = l then   {es exacto}
                        dir := actual;    {lo asigna}
                        actual := dir^.proxima;   {lo desengancha}
                  else     {es mayor que lo necesario}
                        actual^.largo := actual^.largo - l; {descuenta lo usable}
                        dir := actual + actual^.largo; {asigna lo descontado}
                  endif;
                  return dir;
            else    {el primero no sirve}
                  encontre := FALSE;
                  while actual^.proxima <> NIL and not encontre do
                        if actual^.proxima^.largo >= l then {el siguiente nodo
                                                            sirve}
                              if actual^.proxima^.largo = l then  {es exacto}
                                    dir := actual^.proxima;    {lo asigna}
                                    actual^.proxima := actual^.proxima^.proxima;
                                                            {lo desengancha}
                              else    {es mayor que lo necesario}
                                    actual^.proxima^.largo:= actual^.proxima^.largo - l;
                                                            {descuenta lo usable}
                                    dir := actual^.proxima + actual^.proxima^.largo;
                                                            {asigna lo descontado}
                              endif;
                              encontre := TRUE;
                        else   {el siguiente nodo no sirve}
                              actual := actual^.proxima;  {pasa al otro}
                        endif;
                  endwhile;
                  if encontre then
                        return dir;
                  else
```

```
                        return -1;
                  endif;
            endif;
      else   {no hay ningún segmento libre}
            return -1;
      endif;
End Function


c)

Function Instancio_segmento (Largo:long) : Descriptor_Segmento

      Dir = FirstFit (Largo)
      If Dir = -1    { No encontré un segmento libre suficientemente grande }
            Libre = Compactar ()
            If Libre < l  { Aunque compacte, no tengo espacio suficiente }
                  Reemplazar (Largo – Libre) { Debo liberar lo que me falta }
                  Compactar ()     { Compacto para asegurarme que el espacio sea
                                     contiguo }
            Fin
            Dir = FirstFit (Largo)  { Ahora estoy seguro que tengo espacio
                                      suficiente }
      Fin
      Desc = New(Descriptor_Segmento)
      Desc^.Direccion = Dir
      Desc^.Residente = True
      Desc^.Largo = Largo
      return Desc^

End Function
```