Examen Marzo 2008

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

Justifique brevemente cada respuesta.

Conceptos generales

1. Describa brevemente las ventajas/desventajas de los sistemas operativos diseñados con el enfoque monolítico frente al enfoque en capas.
2. Describa dos mecanismos de protección que le brinda el hardware al sistema operativo.

Administración de Procesos

3. Realice un diagrama mostrando los principales campos del descriptor de proceso (*PCB - Process Control Block*) asumiendo que está en un sistema que reconoce hilos a nivel del sistema operativo.
4. ¿Qué funciones desarrolla el despachador (*dispatcher*)?
5. Sean los procesos y tiempos de procesador a utilizar que están en la siguiente Tabla:

Procesos	Tiempo
P_0	5
P_1	8
P_2	7
P_3	9

   Asumiendo un sistema monoprocesador que utiliza un esquema de planificación de tipo *Round-Robin* con *quantum* igual a 3 unidades de procesador.
   a. Dibuje una gráfica de cómo varía en el tiempo el uso del procesador entre los procesos.
   b. Brinde los tiempos de retorno (*turnaround time*) y espera (*waiting time*) de todos los procesos.

Administración de memoria

6. ¿Por cuales etapas debe pasar un código fuente escrito en un lenguaje de programación de alto nivel antes de poder ser ejecutado en un sistema operativo?
7. ¿Qué entiende por asociación de direcciones?

Memoria virtual

8.
   a. ¿Para que son útiles los algoritmos de reemplazo de marcos (*frames*)?
   b. Describa el algoritmo de reemplazo de óptimo (*Optimal Page Replacement*). Muestre sus beneficios y sus limitaciones.

9.
   a. ¿Para qué es útil el modelo del *Working-Set*?.
   b. Describa el modelo del *Working-Set*.

Dispositivos masivos

10. ¿Los sistemas RAID (*Redundant Array of Inexpensive Disk*), a qué mejoras apuntan?

Subsistema de Entrada/Salida

11. Describa el método de Entrada/Salida Programada (*Programmed I/O*). Discuta sus ventajas y desventajas.
12. ¿Qué servicios brinda el núcleo del sistema operativo para el manejo de Entrada/Salida?

Problema 2

Se desea construir un linker que deberá leer un archivo que contiene uno o más módulos, cada uno formado de la siguiente manera:

Tipo de registro	Cantidad	Contenido
`Header`	1	nombre del módulo
`Extern`	0 a n	un símbolo y una dirección
`Public`	0 a m	un símbolo y una dirección
`Código`	1 a l	un largo y un código
`Final`	1	un símbolo

Al final del archivo hay un registro con el siguiente formato:

`final_archivo`	1	un símbolo

- Cada registro `extern` apunta a una palabra (relativa al comienzo del módulo) que deberá contener la dirección de un símbolo definido en otro módulo como `public`.
- Cada registro `public` define la dirección de un símbolo, relativa al comienzo del módulo.
- En cada registro `código` se encuentra una porción del código correspondiente al módulo y el largo del mismo.
- El registro final marca el fin del módulo.
- El registro de tipo `final_archivo` indica que la ejecución de este programa debe comenzar en la dirección de carga de este símbolo.

Suponiendo que la salida del linker va a un cargador (loader) dinámico, con direccionamiento por hardware

- Defina la estructura que el linker debe entregar al cargador (loader).
- Implemente el linker.
- Describa brevemente como es el mecanismo de carga (loading) para este caso.

Justifique todas sus respuestas.

Solución:

a)

La salida del linker será un archivo con todos los registros de código concatenados y las direcciones resueltas.

Por otro lado se debe resolver el simbolo que se encuentra en el registro final_archivo resuelta que es la dirección donde comeienza a ejecutar el problema (símbolo "main").

b)

Como la salida del linker va a un cargador dinámico, con direccionamiento por hardware el linker será:

Recorre todos los registros Extern y forma una tabla con ellos.

Recorre todos los registros Public de los módulos y forma una tabla con ellos.

Forma un archivo de código con todos los registros de código.

Recorre la tabla Extern formada, busca en la tabla Public la dirección que corresponde a cada símbolo y sustituye en el archivo de código.

```
procedure LINKER (entrada: arch_linker, var salida: arch_load var
dir_carga : dirección;
type simb : record
simbolo : char;
direccion : tipo_dir;
sig : ^simb;
end;
var extern,public : ^simb;
dir_act, dir_inicial : tipo_dir;

begin
      dir_act := 0;
      Para cada modulo de entrada, mod_ent
            dir_inicial := dir_act
            Para cada registro Extern de mod_ent, reg_ext
                  Agregar(extern, reg_ext.simbolo, reg_ext.direccion +
                  dir_inicial)
                  {agrega a la lista extern un registro con los valores
                  indicados}
                  Fin Para
            Para cada registro Public de mod_ent, reg_pub
                  Agregar(Public,reg_pub.simbolo,reg_pub.direccion + dir_inicial)
                  {agrega a la lista public un registro con los valores indicados}
            Fin Para
```

```
            Para cada registro Codigo de mod_end, reg_cod
                  Agregar(Salida,reg_cod.codigo)
                  {agrega al archivo de salida el codigo}
                  dir_act := dir_act + reg_cod.largo
                  {acumulo para proxima dir_inicial del siguiente modulo}
            Fin Para
      Fin Para;
      {Ahora viene el registro de "final de archivo" que es el símbolo por
      el cual comienza la ejecución del programa}
      simbolo_main = reg.simbolo
      Para cada elemento de la lista Extern, nodo_ext
            buscar en lista Public el símbolo
            res := find (Public, nodo_ext.simbolo)
            Si res == NULL then
                  { si no existe => error }
                  Print ("Símbolo " , nodo_ext.simbolo, " no encontrado")
                  Exit (1)
            Sino
                  {sino=>ir a la dirección apuntada y reemplazar por la dirección
                  de lista Public encontrada}
                  Memoria [nodo_ext.direccion] := res->direccion
            Fin si
      Fin para
      {Vamos a buscar la dirección apuntada por el símbolo "simbolo_main"}
      res = find (Public, simbolo_main) { suponemos que existe }
      dir_carga = res.direccion
End; {fin LINKER}
```

Problema 3

Una fábrica textil dispone de una máquina con seis motores de hilado cada uno de los cuales utiliza dos tipos diferentes de hilo ubicado en los slots A y B. La máquina cuenta con seis bobinas de hilo de diferentes colores (blanco, negro, rojo, verde, azul y amarillo) que pueden ser usadas por uno solo de los motores al mismo tiempo.

La máquina está configurada en este momento para que el motor uno teja usando hilo blanco (slot A) y negro (slot B), el motor dos use negro (slot A) y rojo (slot B), el motor tres usa rojo (slot A) y verde (slot B), el cuarto use verde (slot A) y azul (slot B), el quinto use azul (slot A) y amarillo (slot B), y el sexto use amarillo (slot A) y blanco (slot B). Cada motor necesita disponer de los dos tipos de hilo al mismo tiempo para poder tejer colocando primero el hilo del slot A y luego el del slot B.

Una vez terminada cada prenda cada motor deberá colocarla en un contenedor común a todos los motores. El mismo puede almacenar hasta 100 prendas en espera de empacar. Por limitaciones físicas del contenedor, solo un motor por vez puede depositar una prenda en el mismo.

Una maquina de empaque es la encargada de ir empaquetando las prendas que se encuentran en el contenedor de a una en orden de llegada.

Se dispone de las siguientes funciones ejecutadas por los motores

- enhebrarHiloA(c: Color)
- enhebrarHiloB(c: Color)
- tejer()
- depositarPrenda()

Se dispone de las siguientes funciones ejecutadas por la empaquetadora

- tomarPrenda()
- empaquetar()

Se pide:

Modelar en ADA las tareas motor y empaquetadora. Se permite utilizar hasta 2 tareas auxiliares.

Solución:

```
ENUM Color(Blanco, Negro, Rojo, Verde, Azul, Amarillo); -- Color va de 1 a 6

TASK Type Motor IS
      ENTRY Colores(Color1:IN Color, Color2:IN Color);
      ENTRY puedeDepositar();
END Motor;

TASK BODY Motor IS
col1, col2:Color;
numMotor :INTEGER;
BEGIN
      ACCEPT Colores(c1, c2)
```

```
            col1 = c1;
            col2 = c2;
      END;
      LOOP
            AdminHilos.entrar();
            AdminHilos.tomar(col1)();
            enhebrarHiloA(col1);
            AdminHilos.tomar(col2)();
            enhebrarHiloB(col2);
            tejer();
            AdminHilos.salir(col1, col2);

            AdminPrendas.inicioDepositar();
            depositarPrenda();
            AdminPrendas.finDepositar();
      ENDLOOP;
END Motor;

Motores : array(1..6) of Motor;

TASK Empaquetadora IS
END Empaquetadora;

TASK BODY Empaquetadora IS
BEGIN
      LOOP
            AdminPrendas.tomarPrenda();
            tomar_prenda();
            AdminPrendas.finTomarPrenda();
            empaquetar();
      ENDLOOP;
END Empaquetadora;


TASK AdminHilos IS
      ENTRY entrar();
      ENTRY tomar(1..6)();
      ENTRY salir(color1: IN Color, color2: IN Color);
END AdminHilos;


TASK BODY AdminHilos IS
```

```
cant: INTEGER;
libre: ARRAY (1..6) of BOOLEAN;
BEGIN
      cant = 0;
      FOR i = 1 to 6
            libre(i) = TRUE;
      END
      Motores(1).Colores(Blanco, Negro);
      Motores(2).Colores(Negro, Rojo);
      Motores(3).Colores(Rojo, Verde);
      Motores(4).Colores(Verde, Azul);
      Motores(5).Colores(Azul, Amarillo);
      Motores(6).Colores(Amarillo, Blanco);
      LOOP
            SELECT
                  WHEN cant < 5 ==>
                        ACCEPT entrar();
                        cant = cant + 1;
                  OR
                        ACCEPT salir(color1: Color, color2: Color)
                              libre(color1) = TRUE;
                              libre(color2) = TRUE;
                        END
                        cant = cant - 1;
                  OR WHEN libre(1) ==>
                        ACCEPT tomar(1)();
                        libre(1) = FALSE;
                  OR WHEN libre(2) ==>
                        ACCEPT tomar(2)();
                        libre(2) = FALSE;
                  OR WHEN libre(3) ==>
                        ACCEPT tomar(3)();
                        libre(3) = FALSE;
                  OR WHEN libre(4) ==>
                        ACCEPT tomar(4)();
                        libre(4) = FALSE;
                  OR WHEN libre(5) ==>
                        ACCEPT tomar(5)();
                        libre(5) = FALSE;
                  OR WHEN libre(6) ==>
                        ACCEPT tomar(6)();
                        libre(6) = FALSE;
```

```
      ENDLOOP
END AdminHilos;

TASK AdminPrendas IS
      ENTRY inicioDepositar();
      ENTRY finDepositar();
      ENTRY tomarPrenda();
      ENTRY finTomarPrenda();
END AdminPrendas;

TASK BODY AdminPrendas IS
prendas: INTEGER;
enUso: BOOLEAN;
BEGIN
      prendas = 0;
      enUso = FALSE;
      LOOP
            SELECT WHEN prendas < 100 AND NOT(enUso) ==>
                  ACCEPT inicioDepositar();
                        enUso = TRUE;
                        prendas = prendas + 1;
            OR ACCEPT finDepositar();
                  enUso = FALSE;
            OR WHEN prendas > 0 ==>
                  ACCEPT tomarPrenda();
            OR ACCEPT finTomarPrenda();
                  prendas = prendas - 1;
      ENDLOOP
END AdminPrendas;
```