Examen Marzo de 2012

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.**
- **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 mínimo de 60 puntos.**

Finalización

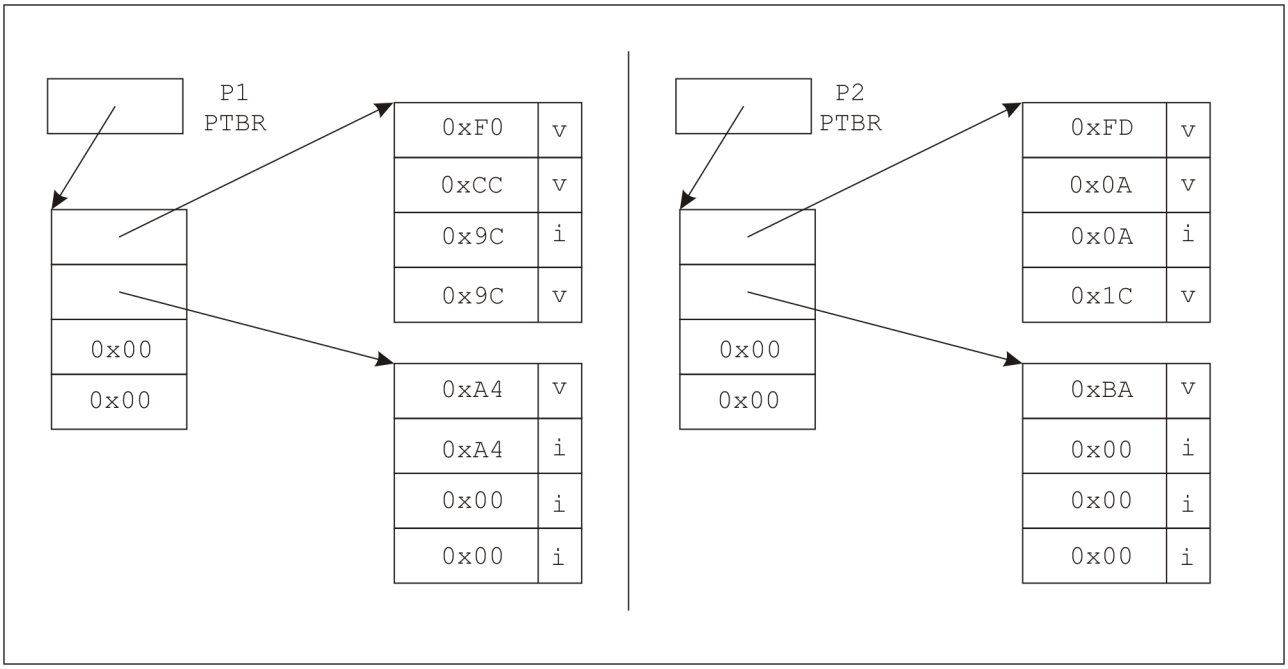
- **El examen dura 4 horas.**

Problema 1 (32 puntos)

1. Qué problema presenta un sistema que no sea multiprogramado?
2. Realice un esquema donde muestre el PCB (*Process Control Block*) de dos hilos (*threads*) de un mismo proceso. Detalle los campos que considere importantes.
3. Realice un diagrama de los estados y sus transiciones que los procesos tienen dentro del sistema operativos. Describa brevemente cada componente.
4. Qué entiende por planificador expropiativo (*preemptive scheduler*)? Qué problema resuelve este tipo de planificador?
5. Por cuáles etapas debe pasar un código fuente escrito en lenguaje C hasta convertirse en un proceso de un sistema operativo?
6. Cuál método de asignación tiene un sistema de archivos FAT (*File Allocation Table*)? Y cuál tiene un sistema de archivos con inodos?
7. Describa los campos principales de una estructura de datos para representar un archivo en una FAT.
8. Describa los métodos de Entrada/Salida Programada y con interrupciones y mencione la ventaja que presenta el método con interrupciones sobre el programado.

Problema 2 (33 puntos)

Sea un sistema con memoria virtual con direcciones virtuales de 12 bits y que está implementado a través de paginación bajo demanda. En este sistema se utiliza un esquema de traducción de dos niveles. Sean la siguientes tablas de páginas para los procesos P1 y P2:



Nota: Las direcciones `0x00` representan memoria virtual no utilizada por el proceso.

1. Determine el tamaño en bytes de las páginas y marcos. (3 pts.).
2. Determine el tamaño máximo del espacio virtual en bytes de este sistema. (3 pts.).
3. Determine el espacio virtual utilizado y la memoria residente en bytes para los procesos P1 y P2 según las tablas de páginas presentadas en la figura. (6 pts.).
4. Suponiendo que el sistema operativo tiene una asignación local de memoria de 4 marcos con un algoritmo de reemplazo LRU (*least recently used*) y sea la siguiente secuencia de accesos a memoria por parte del proceso P1 y P2 (16 pts.):

   ```
   P1:  0 | 0 | 2, 0 | 1 | 6, 0 | 3 | 10, 1 | 0 | 10, 0 | 2 | 3, 1 | 0 | 8,
        0 | 3 | 70, 1 | 1 | 52, 1 | 0 | 10, 0 | 1 | 8
   P2:  0 | 1 | 10, 0 | 0 | 3, 1 | 1 | 10
   ```

   a. Muestre un esquema con el estado de las tablas de página al producirse cada acceso.

   b. Determine cuántos fallos de página se producen.

   Notas:

   - X | Y | Z representa: *X* referencia sobre la tabla de primer nivel, *Y* referencia sobre la tabla de segundo nivel, *Z* desplazamiento.
   - Los cuatro marcos asignados son fijos para cada proceso.

5. Si el proceso P1 realiza un pedido de memoria: `arreglo = (char *) malloc(512)`

   Realice un esquema de las tablas de páginas del proceso P1. (5 pts.).

   Nota: Asuma que el malloc, aparte de pedir memoria, es tomado por el sistema como un acceso a memoria. Partir del estado dejado luego de los accesos de la parte 4.

Solución:

1. Las direcciones virtuales son de 12 bits y se componen de dos niveles de tablas de páginas.

   Como la tabla de páginas de primer y segundo nivel son de 4 entradas, entonces se necesitan 2 bits para direccionar en cada nivel y quedan 8 bits para el desplazamiento sobre la página.

   De esa forma, el tamaño de la página es de 2^8 = 256 bytes.

   Por definición los marcos son del mismo tamaño que las páginas en paginación bajo demanda.

2. Cada proceso tendrá 4 tablas de segundo nivel como máximo y cada una de esas tablas permiten acceder a 4 páginas. Por lo que el espacio máximo de direccionamiento viene dado por:

   4 * 4 * 256 = 2^{12} = 4096 bytes.

3. El espacio virtual utilizado por P1 es de 6 páginas (1536 bytes), mientras que la memoria residente (espacio en memoria física utilizado por el proceso) es de 4 marcos (1024 bytes).

   El espacio virtual utilizado por P1 es de 5 páginas (1280 bytes), mientras que la memoria residente (espacio en memoria física utilizado por el proceso) es de 4 marcos (1024 bytes).

4. a. P1

   Accesos: 0 | 0 | 2, 0 | 1 | 6, 0 | 3 | 10, 1 | 0 | 10

   El estado de las tablas de páginas se mantiene igual.

   LRU queda (0|0, 0|1, 0|3, 1|0).

   Acceso: 0 | 2 | 3 (fallo)

   Tabla 2do nivel 0

0xF0	**i**
0xCC	v
0xF0	**v**
0x9C	v

   Tabla 2do nivel 1

0xA4	v
0xA4	i
0x00	i
0x00	i

   LRU queda (0|1, 0|3, 1|0, 0|2).

   Acceso: 1 | 0 | 8

   LRU queda (0|1, 0|3, 0|2, 1|0).

   Acceso: 0 | 3 | 70

   LRU queda (0|1, 0|2, 1|0, 0|3).

Acceso: 1 | 1 | 52 (fallo)

Tabla 2do nivel 0		Tabla 2do nivel 1	
0xF0	i	0xA4	v
0xCC	**i**	**0xCC**	**v**
0xF0	v	0x00	i
0x9C	v	0x00	i

LRU queda (0|2, 1|0, 0|3, 1|1).

Acceso: 1 | 0 | 10

LRU queda (0|2, 0|3, 1|1, 1|0).

Acceso: 0 | 1 | 8 (fallo)

Tabla 2do nivel 0		Tabla 2do nivel 1	
0xF0	**v**	0xA4	v
0xCC	i	0xCC	v
0xF0	**i**	0x00	i
0x9C	v	0x00	i

LRU queda (0|3, 1|1, 1|0, 0|1).

a. P2

Accesos: 0 | 1 | 10, 0 | 0 | 3

El estado de las tablas de páginas se mantiene igual.

Accesso 1 | 1 | 10

Esto es una dirección virtual válida, pero el proceso no tiene asignado el espacio virtual que quiere acceder por lo que el sistema dará un fallo de direccionamiento. Un ejemplo de este tipo de error es cuando se direcciona fuera de un arreglo (ej.: acceder a arreglo[1000] al arreglo definido en la parte 5).

b. P1 tuvo 3 fallos de página.

5. Al pedir una memoria 512 bytes se necesitan dos nuevas páginas. Una posible asignación es utilzar las páginas 1|2 y 1|3.

   Luego de realizar los accesos se el siguiente estado de las tablas de páginas para el proceso P1:

Tabla 2do nivel 0		Tabla 2do nivel 1	
0xF0	v	0xA4	v
0xCC	i	0xCC	**i**
0xF0	i	**0x9C**	**v**
0x9C	**i**	**0xCC**	**v**

Problema 3 (35 puntos)

Se desea modelar utilizando **monitores** un patio de un complejo habitacional. El patio es usado por perros, gatos y por los vecinos del complejo que toman sol y riegan las plantas.

Los vecinos usan el patio para:

- Tomar sol (siempre que no haya otro vecino tomando sol ni animales en el patio).
- Regar las plantas para lo cual hay una sola manguera. Se pueden regar las plantas incluso con vecinos tomando sol y animales en el patio.

Los gatos y los perros no pueden usar el patio a la vez. Los gatos juegan por dos minutos y luego observan si hay algún perro esperando para usar el patio o algún vecino que quiera tomar sol, de ser así dejan el patio libre. De la misma forma no entrarán nuevos gatos al patio si hay perros esperando para entrar o vecinos que quieran tomar sol.

Se deben implementar los procesos gato(), perro() y vecino().

Notas:

- No se pueden usar tareas auxiliares.
- Los vecinos no podrán llamar a más de dos procedimientos distintos de monitor y no podrán pasar parámetros en ningún caso.
- Los vecinos que quieren tomar sol tendrán prioridad sobre los animales que desean entrar al patio.
- Se dispone de los siguientes procedimientos auxiliares:
  - JugarGato() Invocado por los gatos para jugar por dos minutos en el patio.
  - JugarPerro() Invocado por los perros para jugar en el patio.
  - TomarSol() Invocado por los vecinos para tomar sol.
  - RegarPlantas() Invocado por los vecinos para regar las plantas.
  - QueHacer(): {Sol, Plantas} Invocado por los vecinos para saber que tienen que hacer.

Solución:

```
Monitor PatioJuego
      Condition esperarGaro;
      Condition esperarPerro;
      Condition esperarVecinoTomarSol;

      Integer cantPerrosJugando = 0;
      Integer cantPerrosEsperando = 0;
      Integer cantGatosJugando = 0;
      Integer cantVecinosEsperandoTomarSol = 0;

      procedure entrarGato()
      begin
            if(cantPerrosJugando > 0 OR cantPerrosEsperando > 0 OR
                                        cantVecinosEsperandoTomarSol > 0) then
                  esperarGato.wait();
            end if;
            cantGatosJugando := cantGatosJugando + 1;
            esperarGato.signal();
      end;


      procedure gatoDebeSalir():boolean
      begin
            if(cantPerrosEsperando > 0 or cantVecinosEsperandoTomarSol > 0)
                  cantGatosJugando := cantGatosJugando - 1;
                  if(cantGatosJugando = 0) then
                    if(cantVecinosEsperandoTomarSol > 0) then
                        esperarVecinoTomarSol.signal();
                    else
                        esperarPerro.signal();
                    end if;
                  end if;
                  return true;
            else
                  return false;
            endif
      end;

      procedure entrarPerro()
      begin
            if(cantGatosJugando > 0 or cantVecinosEsperandoTomarSol > 0)
                  cantPerrosEsperando := cantPerrosEsperando + 1;
                  esperarPerro.wait();
                  cantPerrosEsperando := cantPerrosEsperando - 1;
            end if;
            cantPerrosJugando := cantPerrosJugando + 1;
            esperarPerro.signal();
      end

      procedure salirPerro()
      begin
            cantPerrosJugando := cantPerrosJugando - 1;
            if(cantPerrosJugando = 0)
              if(cantVecinosEsperandoTomarSol > 0)
                  esperarVecinoTomarSol.signal();
              else
                  esperarGato.signal();
              endif
            endif
      end;
```

```
      procedure entrarTomarSol()
      begin
            if(cantPerrosJugando > 0 or cantGatosJugando > 0)
              cantVecinosEsperandoTomarSol = cantVecinosEsperandoTomarSol + 1;
              esperarVecinoTomarSol.wait();
              cantVecinosEsperandoTomarSol = cantVecinosEsperandoTomarSol - 1;
            end if;
            TomarSol();
            if(cantVecinosEsperandoTomarSol > 0)
                  esperarVecinoTomarSol.signal();
            else if(cantPerrosEsperando > 0)
              esperarPerro.signal();
            else esperarGato.signal();
      end

end PatioJuego;

Monitor PatioJuegoVecino
      procedure EntrarRegarPlantas()
      begin
            RegarPlantas();
      end

end PatioJuegoVecino;

procedure gato()
      var boolean esperaPerro;
begin
      PatioJuego.entrarGato();
      do
            JugarGato();
      while (not PatioJuego.gatoDebeSalir())
end

procedure perro()
begin
      PatioJuego.entrarPerro();
      JugarPerro();
      PatioJuego.salirPerro();
end

procedure vecino()
begin
      if(QueHacer() = Sol) then
            PatioJuego.EntrarTomarSol();
      else
            PatioJuegoVecino.EntrarRegarPlantas();
      end
end

main()
begin
  cobegin
    perro(); ... perro();
    gato(); ... gato();
    vecino(); ... vecino();
  coend
end
```