Examen 16 de febrero de 2017

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.**
- **Al momento de finalizar el examen no se podrá escribir absolutamente nada en las hojas, debiéndose parar e ir a la fila de entrega. Identificar cada una de las hojas con nombre, cédula y numeración forma parte de la duración del examen.**

Problema 1 (32 puntos)

1. Justifique cuáles de los siguientes algoritmos de planificación puede provocar inanición y cuáles no:
    a. First come, frist serverd (FCFS)
    b. Shortest Job First (SJF)
    c. Round Robin (RR)
    d. Prioridad
2. Considere los siguientes componentes del estado de un proceso: Stack, Heap, Registros del CPU, Variables globales. Cuáles de estos componentes son compartidos entre:
    a. Un proceso padre y su hijo luego de utilizar la operación fork().
    b. Todos los threads de un mismo proceso.
3. Describa tres métodos de planificación de disco.
4. Describa dos servicios que brinda el subsistema de Entrada/Salida.
5. Explique cómo funciona el método de asignación indexada en un sistema de archivos.
6.
    a. Describa y compare los hipervisores de tipo I y II
    b. Indique ventajas y desventajas del uso de máquinas virtuales

7. Explique qué es la hiperpaginación (Thrashing) y describa un método para detectarlo
8. Describa el nivel 5 de RAID.

Problema 2 (34 puntos)

Sea un sistema operativo que utiliza un gestor de memoria el cual implementa memoria virtual utilizando un modelo de paginación bajo demanda.

Se sabe que:

- Las direcciones virtuales son de 56 bits.
- La traducción se realiza a través de 3 niveles de tabla de página.
- Las entradas en las tablas de páginas son de 128 bits (16 bytes).
- Cuando la tabla de 3er nivel esta completa direcciona 64MB (2^26 bytes).
- Las tablas de 2do y 3er nivel ocupan una página completa cada una.

Se pide (justifique cada respuesta):

1. ¿Cuál es el tamaño de las páginas del sistema?

2. Determine cuántos bits son utilizados para determinar el desplazamiento (offset) y cuántos bits para determinar las entradas en la tabla de primer, segundo y tercer nivel.

3. Realice un diagrama que muestre como se realiza la traducción la siguiente dirección virtual:

144	22	18	17

4. Asumiendo que tenemos un proceso que requiere de 600MB para almacenar su código, datos globales, datos dinámicos y que la memoria requerida por su pila es de 40MB
Las direcciones a partir de la dirección virtual 0 se utilizan para almacenar el área de código, el área de datos globales y el área de memoria dinámica.

a) Determine cuántas páginas de las tablas de segundo y tercer nivel son necesarias para poder representar la memoria usada por el proceso.

b) Determine todas las entradas utilizadas en la tabla de primer nivel.

c) Indique el tamaño de la estructura necesaria para soportar el correcto funcionamiento del proceso y compare el tamaño de la tabla de paginas si solamente se tuviera una tabla de páginas de un nivel.

Solución:

1)

Cuando la tabla de 3er nivel está completa direcciona 64MB y en ese caso ocupa una página completa.

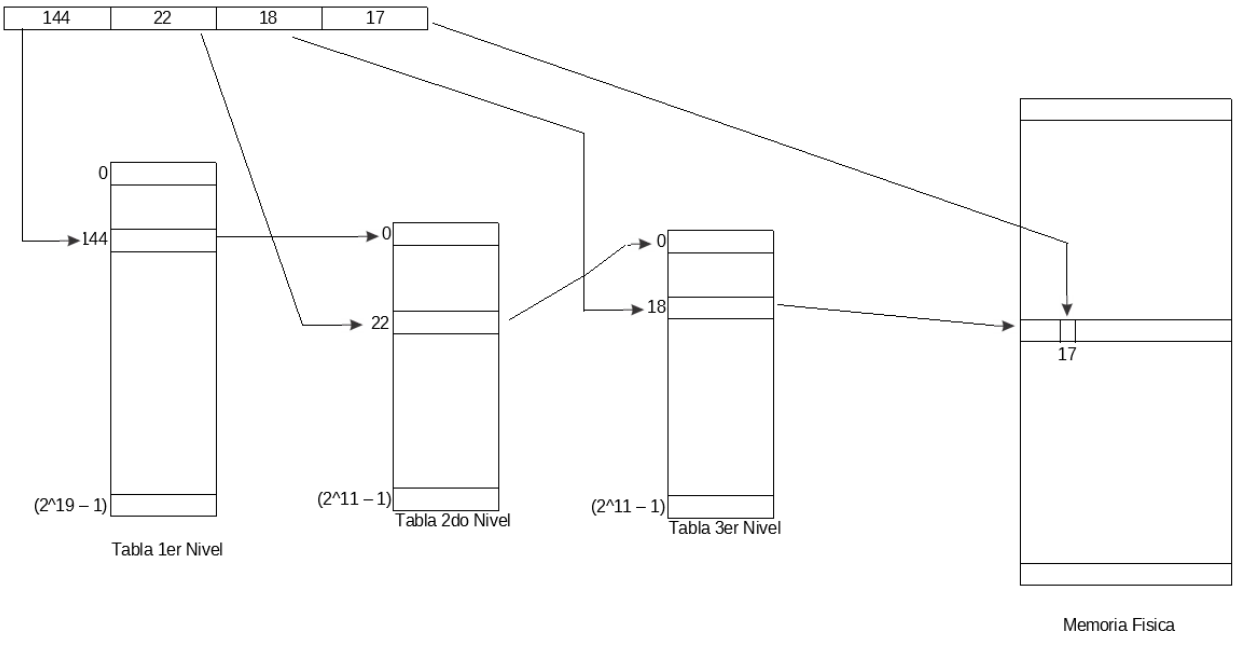
Sea P el tamaño de la página. Cómo las entradas ocupan 16 bytes la tabla de 3er nivel tiene P/16 entradas, cada una apuntando a una página de P bytes.

Por lo tanto (P/2^4) * P = 2^26bytes ==> P = (2^30)^(1/2) ==> P = 2^15 bytes.

2) Cómo las páginas son de 2^15 bytes, se precisan 15 bits para indicar el desplazamiento. Además como la letra dice que las tablas de 2do y 3er nivel ocupan una página completa cada una, se tiene que la cantidad de entradas de las tablas de 2do y 3er nivel son 2^15 bytes/16, es decir 2^11 entradas por lo que se precisan 11 bits para direccionar una entrada del 2do y 3er nivel.

Por último, las direcciones virtuales son de 56 bits => se precisan 56 – 11 – 11 – 15 = 19 bits para direccionar una entrada del 1er nivel.

3)



4)
a) El mapa de memoria para el proceso precisa utilizar 600Mb desde la dirección virtual 0 hacia direcciones crecientes y 40Mb desde la última dirección virtual hacia abajo.
Para el código y datos globales y dinámicos, recordando que cada tabla del tercer nivel mapea 64Mb se precisan 9 páginas completas del tercer nivel más una página parcialmente utilizada.
Para el stack se precisa una tabla parcialmente completa.
Por lo tanto en total se necesitan 11 páginas de tercer nivel.

Como cada tabla del segundo nivel mapea 2^11 * 2^11 * 2^15 bytes = 2^37 bytes = 128Gb se precisa utilizar 1 tabla del segundo nivel para mapear las tablas de tercer nivel de código y dato y otra tabla para mapear la tabla de tercer nivel utilizada por el stack.
Por lo tanto en total se necesitan 2 páginas de segundo nivel.

b)
Un análisis similar al realizado en la parte a) determina que cada entrada de la tabla de primer nivel direcciona 128Gb. Por lo tanto se precisan 2 entradas en la tabla de primer nivel: La entrada 0 y la última entrada (2^19 - 1)

c)
Se precisa tener la tabla de primer nivel más las tablas utilizadas de 2do y 3er nivel.
La tabla de primer nivel ocupa 2^19entradas * 16bytes/entrada = 2^23 bytes = 8Mb
Se utilizan 11 páginas de tercer nivel y 2 páginas de 2do nivel, cada una ocupa 32Kb ==> 416Kb
En total se precisan 8Mb + 416Kb = 8608Kb

Si solamente se tuviese un nivel en la tabla de páginas ésta precisaría 2^56/2^15 entradas, cada una de 16 bytes. Es decir 2^45 bytes para almacenar la tabla.

Problema 3 (34 puntos)

Se desea modelar la playa de contenedores de un puerto sabiendo que por el medio pasa una carretera. Los autos que pasan por la carretera tienen prioridad sobre los camiones que cruzan los contenedores de una sección a otra de la playa. Los camiones cargan un contenedor, cruzan la carretera, descargan el contenedor y vuelven a cruzar la carretera para repetir la tarea. Si la suma de camiones esperando a cruzar la carretera es mayor a 5 estos pasarán a tener prioridad sobre los autos.
Por la carretera también pueden pasar ambulancias las cuales tienen prioridad sobre los camiones en todos los casos.

Se pide:

a) Implementar usando monitores las tareas camión, auto y ambulancia
b) Implementar usando semáforos las tareas camión y auto. No modelar ambulancia.

Se tienen las siguientes funciones auxiliares:

- cruzar(): Invocada por los vehículos para cruzar (autos, camiones o ambulancias)
- cargarContenedor(): Ejecutada por los camiones para cargar un contenedor
- descargarContenedor(): Ejecutada por los camiones para descargar un contenedor del otro lado

Solución:

a)

```
Monitor Cruce
  int esperandoCam, esperandoAuto;
  int autosYambulancias, camiones;
  bool hayAmbulancia;
  Condition esperaAuto, esperaCamion, esperaAmbulancia;

  void inicioAuto()
  {
    if(camiones != 0 || esperandoCam > 5)
    {
      esperandoAuto++;
      esperaAuto.wait();
      esperandoAuto--;
      esperaAuto.signal();
    }
    autosYambulancias++;
  }

  void finAutoOAmbulancia()
  {
    autosYambulancias--;
    if(autosYambulancias == 0)
    {
      esperaCamion.signal();
    }
  }
```

```
  void inicioCamion()
  {
    if(autosYambulancias > 0 || hayAmbulancia ||
       (esperandoAuto > 0 && esperandoCam < 5))
    {
      esperandoCam++;
      esperaCamion.wait();
      esperandoCam--;
      esperaCamion.signal();
    }else
    { // El siguiente signal contempla el caso camiones cruzando,
      // prioridad pasó a autos y vuelve a camiones
      esperaCamion.signal();
    }
    camiones++;
  }

  void finCamion()
  {
    camiones--;
    if(camiones == 0)
    {
      esperaAmbulancia.signal();
      esperaAuto.signal();
    }
  }

  void inicioAmbulancia()
  {
    if(camiones > 0)
    {
      hayAmbulancia = true;
      esperaAmbulancia.wait();
      esperaAmbulancia.signal();
      hayAmbulancia = false;
    }
    autosYambulancias++;
  }

  begin
    esperandoCam = 0; esperandoAuto = 0; autosYambulancias = 0; camiones = 0;
    hayAmbulancia = false;
  end
End;

procedure auto()
{
  Cruce.inicioAuto();
  cruzar();
  Cruce.finAutoOAmbulancia();
}

procedure camion()
{
  while(true)
  {
    cargarContenedor();
    Cruce.inicioCamion();
    cruzar();
    Cruce.finCamion();
```

```
    descargarContenedor();
    // Camion cruza de vuelta para cargar otro contenedor
    Cruce.inicioCamion();
    cruzar();
    Cruce.finCamion();
  }
}

procedure ambulancia()
{
  Cruce.inicioAmbulancia();
  cruzar();
  Cruce.finAutoOAmbulancia();
}

main()
{
  cobegin
    auto; ... auto;
    camion; ... camion;
    ambulancia; ... ambulancia;
  coend
}
```

b)

```
  int autosEsperando, camionesEsperando;
  int autos, camiones;
  Semaphore mutex, semAuto, semCamion;

procedure auto()
{
  P(mutex);
    if(camiones != 0 || camionesEsperando > 5)
    {
      autosEsperando++;
      V(mutex);
      P(semAuto);
      autosEsperando--;
      autos++;
      if(autosEsperando > 0)
        V(semAuto);
      else
        V(mutex); // El P lo hizo camion
    }else
    {
      autos++;
      V(mutex);
    }

  cruzar();

  P(mutex);
    autos--;
    if(autos == 0 && camionesEsperando > 0)
      V(semCamion);
    else V(mutex);
}
```

```
procedure inicio_camion()
{
  P(mutex);
    if(autos > 0 || (autosEsperando > 0 && camionesEsperando < 5))
    {
      camionesEsperando++;
      V(mutex);
      P(semCamion);
      camionesEsperando--;
    }
    camiones++;
    if(camionesEsperando > 0)
      V(semCamion);
    else        // El P lo hizo auto o 6to camion si prioridad pasa a
      V(mutex); // auto mientras pasa camiones y vuelve prioridad a camion
}

procedure fin_camion()
{
  P(mutex);
    camiones--;
    if(camiones == 0)
    {
      if(autosEsperando > 0)
        V(semAuto);
      else V(mutex);
    }else V(mutex);
}

procedure camion()
{
  while(true)
  {
    cargarContenedor();

    inicio_camion();
    cruzar();
    fin_camion();

    descargarContenedor();
    // Camion cruza de vuelta para cargar otro contenedor
    inicio_camion();
    cruzar();
    fin_camion();
  }
}

main()
{
  autosEsperando = 0; camionesEsperando = 0;
  autos = 0; camiones = 0;
  init(mutex, 1);
  init(semAuto, 0);
  init(semCamion,0);
  cobegin
    auto; ... auto;
    camion; ... camion;
  coend
}
```