Examen Diciembre 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 y cada parte del problema de teórico 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 (34 puntos, 3/3/4/4/4/4/4/4/4)

1. Cuál es el principal beneficio de la multiprogramación?
2. Qué es un sistema operativo y qué son los llamados a sistema (system calls)?
3. Describa lo que entiende por sistemas operativos monolíticos y de micro-núcleo.
4. Describa 8 (ocho) campos del bloque descriptor de un proceso (PCB – Process Control Block).
5. Qué entiende por planificador expropiativo (preemptive scheduler)?
6. Describa las 3 (tres) asociaciones de direcciones de memoria vistas en el curso.
7. Describa el problema de hiperpaginación y un método para solucionarlo.
8. Describa 2 (dos) métodos para la administración del espacio libre en un sistema de archivos.
9. Qué beneficios brinda la planificación de disco y describa dos métodos de planificación.

Problema 2 (32 puntos – 2/10/8/12)

Sea un sistema operativo que utiliza un gestor de memoria, el cual implementa memoria virtual utilizando un modelo de paginación bajo demanda. Las direcciones virtuales son de 48 bits y la traducción se realiza a través de 3 niveles de tabla de página. El tamaño de los marcos (*frames*) de la memoria es de 16KB (2^{14} bytes) y se necesitan 128 bits (16 bytes) para identificar inequívocamente a cada uno de ellos.

Notas:

- La tabla correspondiente a el primer nivel solo utiliza una página y la misma siempre esta en memoria.
- Las tablas de página del segundo y tercer nivel tienen el mismo tamaño.
- En las tablas de página solo se almacena los identificadores de marcos.
- 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 (*heap*).
- El resto de las direcciones se destinan al espacio de pila (*stack*), que se almacena comenzando en la dirección virtual más alta y creciendo hacia las direcciones bajas.

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:

768	33	5	20

4. Asumiendo que tenemos un proceso que requiere de 80MB (16KB * 5000) para almacenar su código, datos globales en memoria y datos dinámicos, y que la memoria requerida por su pila es de 20MB (16KB * 1250):

a) Determine cuántas páginas de 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.

Solución:

1. Por definición las páginas y los marcos tienen el mismo tamaño. De esa forma, el tamaño de las paginas de sistema es de 16KB.

2. Por un lado, como las páginas tienen un tamaño de 16KB (2^{14} bytes) son necesarios 14 bits para el desplazamiento.

Por otro lado se sabe que: "En las tablas de página solo se almacena los identificadores de marcos." entonces sabemos que el descriptor almacenado en cada una de las entradas de la tabla de pagina es de 128bits (16bytes) y como "La tabla correspondiente a el primer nivel solo utiliza una página y la misma siempre esta en memoria.". Dividiendo el tamaño de la pagina por el tamaño de descriptor tenemos la cantidad de entradas que tiene la tabla de primer nivel, luego tenemos la cantidad de bits necesarios para indexar cada uno de ellas.

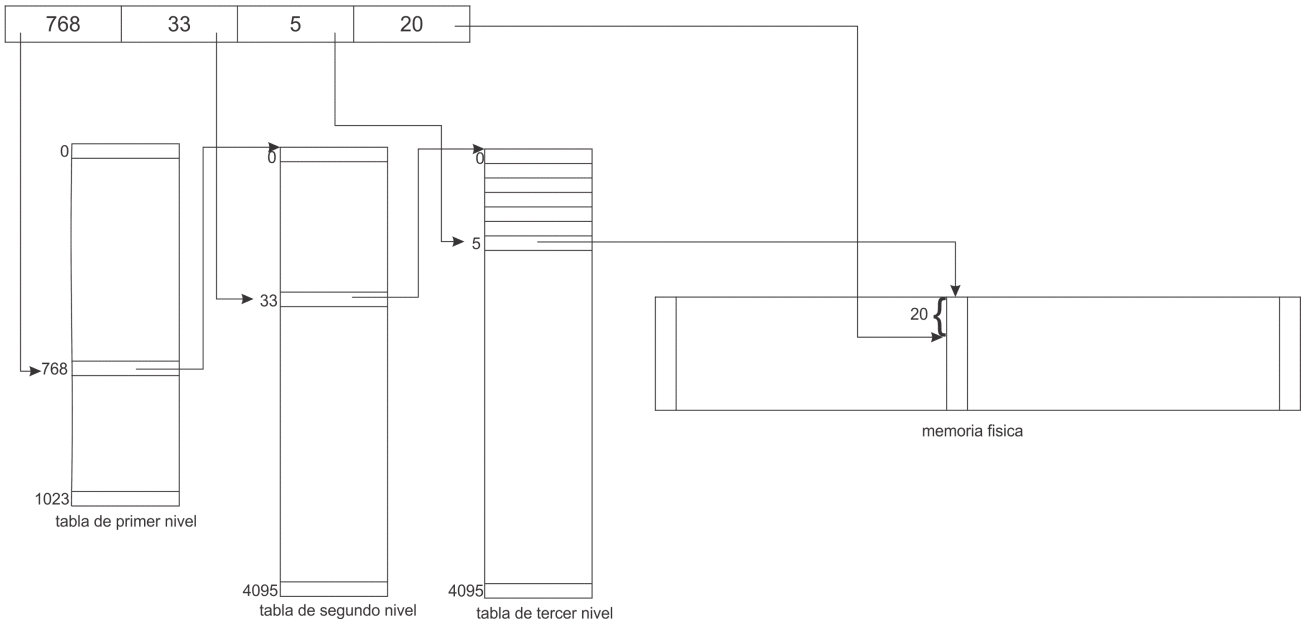
$$\frac{16\,KB}{128\,bits} \Rightarrow \frac{2^{14}\,bytes}{16\,bytes} \Rightarrow \frac{2^{14}\,bytes}{2^{4}\,bytes} \Rightarrow 2^{10} \Rightarrow \begin{array}{l} se\ precisan\ 10\ bits\ para\ indexar\ a\ cada\ una \\ de\ las\ entradas\ de\ la\ tabla\ de\ primer\ nibel. \end{array}$$

Finalmente nos quedan 24 bits (48 – 14 – 10) para indexar en las tablas de segundo y tercer nivel; y como sabemos que "Las tablas de página del segundo y tercer nivel tienen el mismo tamaño." entonces tenemos 12 bits para determinar la entrada utilizada de las tabla segundo nivel y 12 bits para determinar la entrada utilizada en la tabla de tercer nivel.

Resumiendo tenemos que:

- Se precisan 10 bits para determinar la entrada utilizada en la tabla de primer nivel.
- Se precisan 12 bits para determinar la entrada utilizada en la tabla de segundo nivel.
- Se precisan 12 bits para determinar la entrada utilizada en la tabla de tercer nivel.
- Se tiene 10 bits para determinar el desplazamiento.

3.



4.a. Para representar los 80MB (16KB * 5000) de código, datos globales y datos dinámicos se precisan 5000 páginas. Por ende se precisan 5000 entradas en la tabla de tercer nivel, como cada tabla de tercer nivel tiene 4096 (2^{12}) entradas, entonces se requieren de 2 tablas de tercer nivel. De esa forma, se requieren de 2 entradas de la tabla de segundo nivel y como la tabla de segundo nivel tiene 4096 entradas (al igual que la de tercer nivel), entonces es necesaria una tabla de segundo nivel.

Análogamente para representar los 20MB (16KB * 1250) se requieran de 1250 páginas y como la tabla de tercer nivel tiene 4096 entradas una sola tabla de tercer nivel se suficiente. Para almacenar una sola tabla de tercer nivel es necesaria una sola tabla de segundo nivel.

Entonces para representar toda la memoria se requieren de 3 tablas de tercer nivel y 2 tablas de segundo nivel.

Tanto las tablas de segundo como de tercer nivel tienen 2^{12} entradas y un descriptor de 128bits (16bytes o 2^4 bytes), entonces el tamaño de cada tabla es de 64KB (2^{12} * 2^4 bytes). Entonces cada tabla precisa de 4 páginas (64KB / 16KB) para ser almacenada.

Finalmente la cantidad de paginas necesarias para almacenar las tablas de segundo nivel es de 8 páginas y las cantidad de páginas necesarias para almacenar las tablas de tercer nivel es de 12 páginas.

4.b. Por lo visto en el punto 4.a para representar el código, los datos globales y los datos dinámicos se requiere de una sola tabla de segundo nivel, esto implica que se utilizara una entrada de la tabla de primer nivel y como esta sección de memoria comienza en la dirección virtual 0 la entrada de la tabla de primer nivel que se utilizara sera la primera (la entrada número 0).

Por lo visto en el punto 4.b se requiere de una tabla de segundo nivel para representar el stack (espacio de la pila). Esta sección de memoria comienza en la direcciones mas alta y creciendo hacia las direcciones más bajas. Esto implica que se utilizara la ultima entrada (la entrada número 1023) de la tabla de paginas de primer nivel.

Problema 3 (34 puntos)

Se dispone de un dispositivo de hardware capaz de enviar mensajes SMS que cuenta con 5 chips. Cuando el dispositivo acepta una solicitud de envío de SMS a un determinado chip contesta con un mensaje que indica solamente si la solicitud fue aceptada. Debido a los tiempos de validación este mensaje puede demorar y las respuestas pueden llegar en un orden diferente a los pedidos.

Posteriormente, si la solicitud fue aceptada, enviará un segundo mensaje indicando si el envío del SMS al destinatario se logró realizar con éxito. El dispositivo puede hacer varios intentos por lo que este segundo mensaje también puede demorar. Este segundo mensaje incluirá además el número del chip por el que se envió el SMS.

Se desea programar utilizando monitores un servicio de mensajería SMS que utilice este dispositivo. Para ello se usará un proceso servidor que escuchará los mensajes del dispositivo mientras que habrá *N* procesos clientes que enviarán los SMS. Se desea lograr la máxima concurrencia posible en el uso de los 5 chips del dispositivo.

Si no hay chips libres los clientes deberán esperar a que se libere uno. El chip se considera libre luego de recibir el segundo mensaje del dispositivo o luego de recibir el primero si la solicitud no fue aceptada (por ejemplo porque el número del destinatario no es válido). Antes de terminar, los clientes deberán indicar si el SMS fue enviado con éxito o no. Un SMS enviado con éxito es aquel que recibió respuestas positivas en ambos mensajes (si el primer mensaje es fallido el segundo no es enviado por el dispositivo)

Se dispone de los siguientes procedimientos:

• `obtener_SMS(): SMS`	Ejecutado por los clientes para obtener el SMS a enviar.
• `enviar_mensaje(sms: SMS, chip: [1..5])`	Ejecutado por los clientes para enviar el SMS al dispositivo.
• `sms_enviado(ok: Boolean)`	Ejecutado por los clientes para indicar que se pudo mandar el SMS correctamente o que el dispositivo no pudo entregarlo al destinatario.
• `recibir_respuesta(): Respuesta`	Ejecutado por el proceso servidor para recibir los mensajes del dispositivo.

La estructura respuesta tiene este tipo:

```
Respuesta = Record
      Case tipo: Tipo of
            aceptado: (ok: Boolean);
            enviado: (ok: Boolean; chip: [1..5]);
      end;
```

Nota:

- No se pueden usar tareas auxiliares.

Solución:

```
Monitor Control
  Condition esperaLibre, chipEnvio;
  boolean libres[1..5];
  boolean puedoEnviar;
  int cantLibres;
  int chip;

  int ultimoEnvio()
  {
    int ultimoChip = chip;
    puedoEnviar = true;
    esperaEnvio.signal();
    return ultimoChip;
  }

  int quieroEnviar()
  {
    int chipEnvio = 0;
    boolean encontrado = false;

    if(cantLibres == 0)
      esperaLibre.wait();
    cantLibres--;
    while (!encontrado && (++chipEnvio) <= 5)
    {
      if(libres[chipEnvio])
      {
        libres[chipEnvio] = false;
        encontrado = true;
      }
    }
    if(!puedoEnviar)
      esperaEnvio.wait();
    chip = chipEnvio;
    puedoEnviar = false;
    return chipEnvio;
  }

  void prontoEnvio(int chip)
  {
    cantLibres++;
    libres[chip] = true;
    esperaLibre.signal();
  }

  begin
    for(int i = 0; i < 5; i++)
      libres[i] = true;
    cantLibres = 5
    chip = 0
    puedoEnviar = true
  end
EndMonitor
```

```
Type Monitor ChipMon
  Condition esperaRespuesta;
  bool respuestaOk;
  bool enviar(SMS msg, int chip)
  {
    enviar_mensaje(msg, chip);
    esperaRespuesta.wait();
    return respuestaOk;
  }

  void respuesta(boolean ok)
  {
    respuestaOk = ok;
    esperaRespuesta.signal();
  }

  begin
  end
EndMonitor

procedure servidor
{
  while(true)
  {
    Respuesta respuesta = recibir_respuesta();
    if(respuesta.tipo == aceptado)
    {
      int chip = Control.ultimoEnvio();
      if(!respuesta.ok)
        Chip[chip].respuesta(false);
    }else
    {
      Chip[respuesta.chip].respuesta(respuesta.ok);
    }
  }
}

procedure cliente
{
  SMS msg = obtener_SMS();
  int chip = Control.quieroEnviar();
  boolean resultado = Chip[chip].enviar(msg, chip);

  Control.prontoEnvio(chip);
  sms_enviado(resultado);
}

ChipMon Chip[1..5];

begin
  cobegin
    servidor;
    cliente; cliente; ... cliente;
  coend
end
```