Examen Julio de 2009

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 cada respuesta.

1. ¿Qué entiende por sistema operativo ?
2. ¿Qué ventajas presenta un sistema multiprocesador simétrico frente a uno asimétrico ?
3. ¿Por qué considera necesario mecanismos de inter-comunicación entre procesos (IPC) en un sistema operativo ?
4. Sea un sistema monoprocesador con un planificador expropiativo (preemptive) que se ejecuta al comenzar cada unidad de tiempo.

   El planificador utiliza prioridades de 1 a 10 (a menor número mayor prioridad), la cual es decrementada en 1 (uno) cada vez que un proceso está esperando por el recurso procesador y se incrementa en 1 (uno) cada vez que un proceso está ejecutando en un procesador.

   Dada la siguiente tabla de procesos, con sus prioridades, unidades de tiempo restantes por ejecutar y en que unidad de tiempo fue creado:

Proceso	Prioridad	Unidades de tiempo x ejecutar	Tiempo de creación
p1	10	5	0
p2	2	4	3
p3	6	4	2

   a. Realice un esquema en el tiempo sobre como será distribuido el recurso procesador entre los procesos en el tiempo.
   b. ¿ Cuál es el tiempo de espera de cada proceso ?

Nota: En caso de igualdad de prioridades asuma un caso.

5.
   a. ¿ Qué ventajas presenta un sistema operativo que brinde soporte a hilos (threads) a nivel del sistema operativo ?
   b. ¿ En que modelos de hilos es necesario un planificador a nivel de usuario ?

6.
   a. ¿ Qué entiende por espacio virtual asignado a un proceso ?
   b. ¿ Los hilos de ejecución de un proceso tienen acceso ilimitado a este espacio virtual ?

7.
   a. ¿ Para que sirve la TLB (Translation Look-aside Buffer) ?
   b. Defina el tiempo efectivo de acceso (EAT) a este nivel.
   c. ¿ Qué información brinda esta medida ?

8. ¿ Qué ventajas brinda los sistema RAID (Redundant Array of Inexpensive Disk) ?

Solución:

Ej. 4)

```
  p1  p1  p3  p2  p2  p2  p3  p2
+---+---+---+---+---+---+---+---+ …
0   1   2   3   4   5   6   7   8
```

A partir del tiempo 8 tenemos 4 posibles casos:

```
...  p1  p3  p1  p3  p1    TE p1 = 8, TE p3 = 6
+--+---+---+---+---+---+
   8   9  10  11  12  13
o
...  p1  p3  p3  p1  p1    TE p1 = 8, TE p3 = 5
+--+---+---+---+---+---+
   8   9  10  11  12  13
o
...  p3  p1  p3  p1  p1    TE p1 = 8, TE p3 = 5
+--+---+---+---+---+---+
   8   9  10  11  12  13
o
...  p3  p1  p1  p3  p1    TE p1 = 8, TE p3 = 6
+--+---+---+---+---+---+
   8   9  10  11  12  13
```

b. TE p2 = 1

Problema 2

Un sistema de gestión de memoria, soporta espacios de **direcciones lógicas (virtuales) de 32 bits** y un modelo de memoria **paginada tradicional** (tabla de página de un nivel), con tamaños de página de **4KB**.

Consideramos una técnica alternativa para administrar la memoria en este sistema de 32 bits, denominada **partición paginada**. Este esquema consiste en dividir los espacios de direcciones, en **particiones**. Cada **partición** se divide a su vez en páginas de **4KB**. En nuestro caso, **una partición contiene 2048 páginas**.

La memoria principal (física) es un **espacio de direcciones de 63 bits**.

Se pide:

1. Suponiendo que usamos un esquema de memoria paginada tradicional, si se mantuviese completamente la tabla de páginas en memoria principal, indique el tamaño que tendría esta.
2. Suponiendo que usamos el esquema de memoria partición paginada, represente gráficamente las direcciones lógicas utilizadas, indicando todos sus campos. Represente el mecanismo de traducción de direcciones indicando las tablas involucradas, sus tamaños, estructura y tamaño de cada descriptor de dichas tablas. A su vez, se debe indicar quien es el responsable de realizar cada traducción.
3. Estudie la conveniencia de utilizar uno u otro esquema.
4. Tal como se presenta aquí el esquema de partición paginada, indique que diferencias presenta con un esquema de segmentación paginada.

Solución:

a) Paginación tradicional

En un sistema con paginación de memoria tradicional, debemos seguir el siguiente procedimiento para realizar el cálculo:

- Las direcciones tienen 32 bits de largo.
- Los marcos tienen 4kb de tamaño, por lo que necesitamos 12 bits para direccionar dentro del marco.
- Nos quedan 20 bits para direccionar marcos.

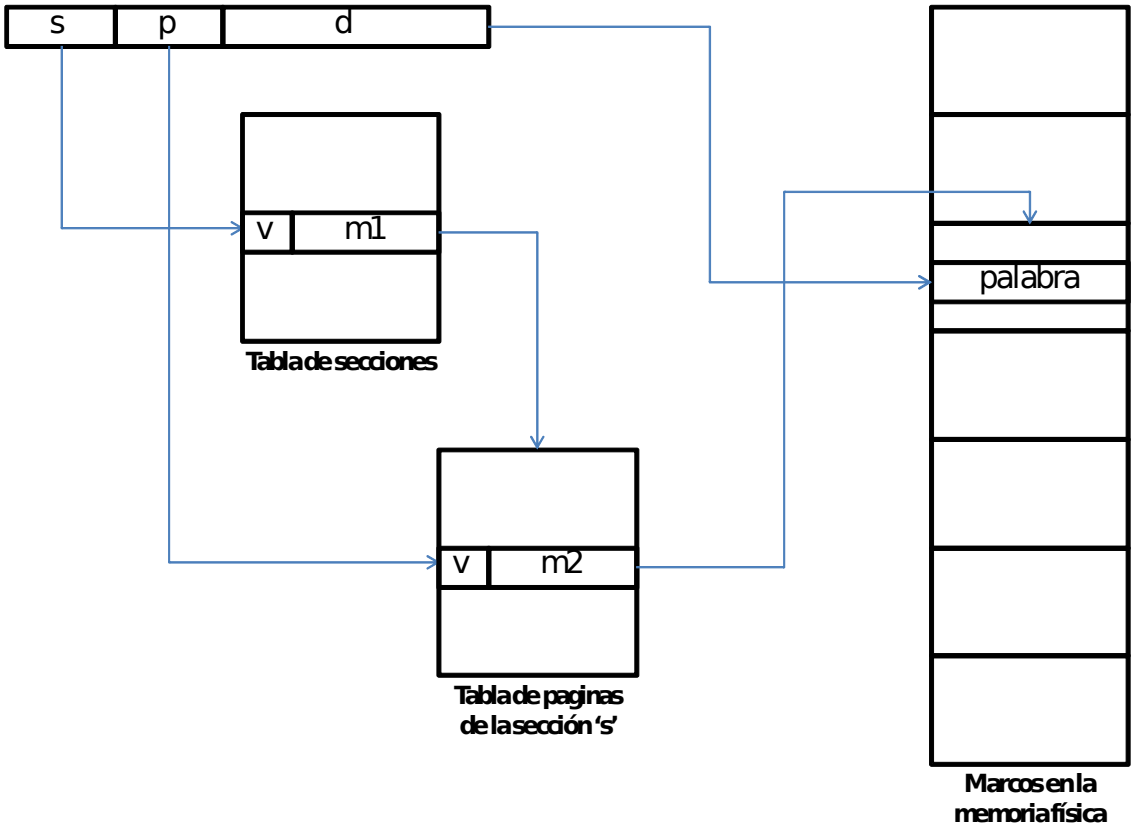
Cada descriptor en la tabla de páginas, debe tener la dirección de un marco. Debido a que el espacio de memoria es de 63 bits, entonces necesitamos 63 bits como mínimo para direccionar en la memoria real. A su vez, agregamos un bit para indicar si la página está presente o no (bit de validez). Por lo tanto, cada descriptor ocupa 64 bits, lo que equivale a 8 bytes.

Tenemos 2^{20} entradas en la tabla de páginas, con lo que el tamaño total de la tabla de páginas es de 2^{20} x 8 bytes.

2^{20} x 8 bytes = 1048576 x 8 bytes = 8388608 bytes = 8 Mb.

b) Partición paginada

En el siguiente diagrama, se presenta el proceso de traducción de una dirección lógica de 32 bits, en una dirección física para acceder a la palabra de memoria buscada. Además el proceso de traducción será realizado por el MMU.



En este caso, el desplazamiento d, es de 12 bits, ya que el marco sigue teniendo 4Kb. Una partición contiene 2048 páginas, por lo que necesitamos 11 bits para direccionar las páginas de una sección.

Esto hace que nos queden 9 bits libres para direccionar secciones. En este esquema, podemos tener como máximo, 512 secciones.

- Cada descriptor de la tabla de secciones tiene 1 bit de validez, y NN bits para direccionar una tabla de páginas de dicha sección.
- Tenemos como máximo 512 secciones y 2048 páginas por sección.
- Cada descriptor contiene un bit de validez, y 63 bits para direccionar una posición en la memoria real. La tabla de páginas, se encuentra en la memoria real. Por lo tanto, cada descriptor ocupa 8 bytes.
- En memoria, tendremos como mínimo la tabla de secciones y una tabla de páginas de la sección seleccionada en memoria.

Entonces, para que el sistema funciones, necesitamos como mínimo 512 x 8 bytes + 2048 x 8 bytes = 2560 x 8 bytes = 20480 bytes = 20 Kb

c) Dependiendo del criterio usado, será conveniente una solución u otra: Como se dijo, si lo que se quiere es ahorrar en memoria usada por las tablas, será más eficiente la solución de partición paginada. Por otra parte, si se quiere tener una solución más rápida (y no se está limitado por la memoria), la partición paginada realiza una lectura más de memoria, y utiliza más lógica en la MMU, lo que hará preferible la opción de paginación de un nivel.

d) El esquema presentado es en realidad una forma diferente de nombrar un esquema de paginación de dos niveles. Sin embargo, comparando el mismo con un sistema de segmentación paginada, este esquema no hace una división lógica de las secciones y no brinda la opción de tener secciones de distintos tamaños, como es el caso con los segmentos.

Problema 3

En vista de los comicios de octubre-noviembre, la corte electoral ha decidido implementar una serie de cambios apuntando a acelerar la atención de los votantes teniendo dos cuartos secretos por mesa de votación.

La mesa de votación atiende a los votantes de a uno por vez. Los votantes deberán entregarle la credencial para que la mesa verifique la validez de la misma. Si la credencial es válida se le pasa un sobre al votante mientras que si no lo es se le comunica este hecho. En caso de que sea válida la mesa anota en la planilla el número de sobre y el número de credencial del votante.

Si obtuvo el sobre se debe dirigir a un cuarto secreto siempre y cuando haya alguno libre. Sino esperará en una fila por orden de llegada.

Una vez en el cuarto secreto, el votante pondrá la hoja de votación en el sobre. Cuando esta tarea finalice, retornará a la mesa, le pasará la tirilla del sobre, recibirá la credencial, quedando entonces habilitado a depositar el voto.

La mesa puede atender más votantes mientras los cuartos secretos estén ocupados o mientras algún votante esté depositando su voto. En el momento que el votante salga del cuarto secreto a entregar su tirilla puede tener que esperar a que la mesa termine la atención de otro votante, pero tendrá prioridad frente a los demás votantes que lleguen a iniciar el trámite.

Se desea modelar utilizando mailboxes, los procesos antes descriptos. (Mesa y Votante).

Puede asumir la existencia de las siguientes funciones:

- **mi_credencial(): credencial**
  Devuelve la credencial del votante que ejecuta la función
- **validar_credencial(credencial): boolean**
  Indica si la credencial dada es válida
- **tomar_sobre(): sobre**
  Toma un sobre
- **registrar_credencial(credencial, sobre)**
  Anota los números de credencial y sobre en la planilla
- **elegir_voto(sobre, nro_cuarto)**
  Utilizada por los votantes para entrar a un cuarto secreto e introducir la lista en el sobre. nro_cuarto es un entero en el rango (1,2)
- **sacar_tirilla(sobre): tirilla**
  Utilizada por los votantes para cortar la tirilla del sobre
- **obtener_credencial(tirilla): credencial**
  Utilizada por la mesa para saber que credencial devolver
- **depositar_voto(sobre)**
  Deposita el voto en la urna.

Solución:

```
Mailbox mb_cuartos of Integer;
Mailbox mb_fila_tirilla of Integer;
Mailbox mb_requiero_atencion of Nil;
Mailbox mb_espero_atencion of Nil;
Mailbox mb_votante_credencial of Credencial;
Mailbox mb_sobre of {Integer, Sobre, Credencial};
Mailbox mb_quiero_poner_sobre of Nil;
Mailbox mb_tirilla of Tirilla;
Mailbox mb_credencial of Credencial;

procedure Mesa
      mb_cuartos.send(1);
      mb_cuartos.send(2);
      mb_fila_tirilla.send(0});

      while(true)
      {
            mb_requiero_atencion.receive();
            mb_fila_tirilla.receive(CantEsperaTirilla);
            if(CantEsperaTirilla == 0)
            { // No hay nadie esperando para entregar tirilla
                  mb_fila_tirilla.send(CantEsperaTirilla);
                  mb_espero_atencion.send(NIL);
                  mb_votante_credencial.receive(Credencial);
                  if(validar_credencial(Credencial))
                  {
                        Sobre = tomar_sobre();
                        registrar_credencial(Credencial, Sobre);
                        mb_sobre.send(VALIDA, Sobre, NIL);
                  }else
                  {
                        mb_sobre.send(INVALIDA, NIL, Credencial);
                  }
            }else
            { // Hay votantes esperando para entregar tirilla
                  mb_fila_tirilla.send(CantEsperaTirilla - 1);
                  mb_quiero_poner_sobre.send(NIL);

                  mb_tirilla.receive(Tirilla);
```

```
                    Credencial = obtener_credencial(Tirilla);
                    mb_credencial.send(Credencial);
              }
        }
end procedure;
procedure Votante
        mb_requiero_atencion.send(NIL);
        mb_espero_atencion.receive();

        mb_votante_credencial.send(mi_crecencial());
        mb_sobre.receive({Resultado, Sobre, Credencial});

        if(Resultado == VALIDA)
        {
              mb_cuartos.receive(NroCuarto);
              elegir_voto(Sobre, NroCuarto);
              mb_cuartos.send(NroCuarto);

              mb_fila_tirilla.receive(CantEsperaTirilla);
              mb_fila_tirilla.send(CantEsperaTirilla + 1); // Quiero entregar
tirilla
              mb_requiero_atencion.send(NIL);
              mb_quiero_poner_sobre.receive(); // Espero atencion de la mesa

              Tirilla = sacar_tirilla(Sobre);
              mb_tirilla.send(Tirilla);
              mb_credencial.receive(Credencial);
              depositar_voto(Sobre);
        }
end procedure;
main()
begin
 cobegin
    Mesa;
    Votante;
      ...
    Votante;
 coend
end;
```