Examen Diciembre de 2003

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

Para las siguientes partes, conteste justificando adecuadamente, cada una de las preguntas.

- **Parte 1**

a. Explique muy brevemente las condiciones necesarias para la existencia del bloqueo mutuo (Deadlock) en sistemas donde exista solo una instancia de cada tipo de recurso. Cuales condiciones debería agregar para que además de necesarias sean suficientes?

==========

Solución:

Las condiciones son (por más detalles ver libro):

*Exclusión mutua

*Retención y espera

*No apropiación

*Espera circular

Si hay una sola instancia de cada tipo de recurso estas condiciones son necesarias y suficientes.

==========

b. Que puede decir de las siguientes situaciones viendo los estados de las variables durante la ejecución del algoritmo del banquero. Justifique la respuesta muy brevemente.

Caso1:

Matriz Asignación

2	1	0
1	2	0
0	2	0

Solicitud proceso 2

0	1	1

Matriz Disponibilidad:

1	2	1

Matriz Máximos:

2	1	1
1	4	1
0	4	0

==========

Solución:

La solicitud puede ser satisfecha manteniendo el sistema en un estado seguro.

==========

Caso 2: Idem caso 1, pero la solicitud para el proceso 2 es :

1	0	1

==========

Solución:

El proceso 2 no puede realizar esa solicitud pues excedería su máximo predefinido.

==========

- **Parte 2**

a. ¿Que es un archivo?, ¿Y un sistema de archivos?

b. Enumere al menos cinco atributos que deban conservarse para describir un archivo.

c. ¿Que es un i-nodo?

==========

Solución:

Ver libro

==========

- **Parte 3**

a. En el diseño de un planificador, explique la diferencia entre apropiativo y no apropiativo

b. Para el diseño de un sistema operativo, señale virtudes y defectos comparando una solución monolítica con una basada en un micro núcleo.

==========

Solución:

Ver libro

==========

Problema 2

Se desea construir el modulo de administración de memoria virtual de un sistema operativo para un sistema de computación con las siguientes características:

- Utiliza paginación de memoria
- Las páginas son de 4K Bytes
- Utiliza una tabla de paginas de como máximo 1M entradas, pudiendo ocupar esta tabla (en su totalidad) como máximo, 4K Bytes
- Una memoria física de 256M Bytes

Nota: Se desea aprovechar lo máximo posible el uso de la memoria.

a) Indique el tamaño y formato de las direcciones lógicas. Indique el formato de las entradas en la tabla de páginas.

==========

Solución:

Páginas de 4kb = 4096 = 2^{12} -> Se requieren 12 bits para direccionar en cada página.

La tabla de paginas puede tener a lo máximo 1K entradas, por lo que tenemos 2^{10} entradas. El formato de la dirección lógica es entonces:

```
   <------- 10 bits -----><------- 12 bits --------->

       Número de página      Offset en la página
```

Como la tabla de páginas puede ocupar como máximo 4Kb, y se desea aprovechar la memoria al máximo, se define que cada entrada en la tabla de páginas ocupe 4bytes, ya que 2^{10} = 4Kb. En cada entrada almacenamos m bits para determinar el numero de marco en la cual esta la página que buscamos. Tenemos 256mb = 2^{28} y marcos de 4K = 2^{12} por lo que tenemos 2^{16} marcos y necesitamos 16 bits para direccionar cada marco (nótese que en este caso no podremos direccionar toda la memoria simultáneamente dado que solamente tenemos 2^{10} entradas en la tabla de páginas).

El formato de cada entrada será:

```
  <---------- 16 bits ----------><---------- 16 bits ---------->

           Número de página                Bits libres
```

Donde los 16 bits que sobran se utilizan para los algoritmos de reemplazo (parte c) y para posibles bits de validez

==========

b) Explique el proceso de traducción de una dirección lógica en una dirección física

==========

Solución:

Cada dirección generada por la CPU se divide en dos partes, un numero de pagina (p) y un desplazamiento en la pagina (d). El número de página se utiliza como índice en una tabla de páginas, que contiene la dirección base para cada página de la memoria física. La dirección base se combina con el desplazamiento dentro

de la página para obtener una dirección física absoluta. El tamaño de la página y del marco están definidos por el hardware.

==========

c) Se desea, sobre el esquema anterior, implementar un algoritmo de reemplazo de páginas con las siguientes propiedades:

- Por cada pagina se tiene un entero sin signo, que representa la utilización de la pagina.
- Cuando la pagina se asigna por primera vez, este entero se pone a cero.
- Cada vez que se accede a la página, se deberá incrementar dicho entero (de alguna forma) a fin de reflejar un aumento en la utilización de la misma.

1. En base a estas premisas como implementaría usted un algoritmo de reemplazo de páginas que elimina las páginas menos usadas.

==========

Solución:

Aprovechando los 16 bits sobrantes, definimos un entero sin signo para representar el tiempo que la página lleva activa. Cuando la página se asigna por primera vez, estos 16 bits se establecen en 0. Cada vez que se consulta la tabla de páginas, tanto para lectura o escritura, se aplican las siguientes operaciones:

1.- xxxx xxxx xxxx xxxx >> 1 = 0xxx xxxx xxxx xxxx

2.- 0xxx xxxx xxxx xxxx | 1000 0000 0000 0000 = 1xxx xxxx xxxx xxxx

Lo que obtenemos en este caso es un numero mayor, que indica que la página sigue en uso.

El algoritmo de reemplazo recorre las entradas en la tabla de páginas para elegir la que tenga el entero menor.

==========

2. Implemente el algoritmo planteado en 1)

==========

Solución: (en pseudocodigo)

El algoritmo está compuesto por varios módulos que ejecutan en distintas ocasiones.

Ante el acceso a una página, se conoce cual es la dirección logica que se está utilizando. Sin importar si el acceso es para lectura o escritura, se debe invocar una rutina con la forma siguiente:

```
// Este procedimiento recibe una dirección lógica y
// actualiza la entrada en la tabla de páginas
procedure actualizarEntradaTabla(int direccion_logica)
begin
   int indice  = direccion_logica / 2^12;
   int entrada = TablaPaginas[indice];
   int conteo  = entrada & 0000 0000 0000 0000 1111 1111 1111 1111
   conteo = conteo >> 1;
   conteo = conteo | 0000 0000 0000 0000 1000 0000 0000 0000
   entrada = entrada | conteo
   TablaPaginas[indice] = entrada;
end;
```

Cuando se produce un fallo de página, y debe seleccionarse una entrada de la página para eliminar, el algoritmo recorre la tabla buscando la que tenga el menor número. La que así sea, es eliminada y enviada a disco.

```
   // actualiza la entrada en la tabla de paginas
   procedure int seleccionarEntradaTabla()
   begin
      int mask = 0000 0000 0000 0000 1111 1111 1111 1111;
      int minIdx = -1;
      int minCtd = INFINITO;
      for (int i = 0; i < 2^10; i++)
            if (TablaPaginas[i] == 0) then
                  return i;
            else
                  int ctd = TablaPaginas[i] & mask;
                  if (ctd < minCtd) then
                        minIdx = i;
                        minCtd = ctd;
                  endif
                  endif
      endfor

      swapearPagina(TablaPaginas[minIdx]);
            limpiarMemoria(TablaPaginas[minIdx]);
      return minIdx;
   end;
```

==========

Problema 3

Sea una maquina receptora de envases que tiene 3 bocas de recepción (boca1, boca2, boca3) y que acepta 5 tipos de envases (1..5). Las bocas deberán poder funcionar concurrentemente.

Los envases deberán ser almacenados en 5 cajones (de 6 envases cada uno) distintos dependiendo de su tipo.

Cuando un cajón se llena, el supervisor deberá ser avisado de este evento para que cambie el cajón correspondiente. Y si alguien introdujera una botella de ese tipo deberá quedarse esperando (esa boca) hasta tanto se cambie dicho cajón (sin poder ingresar ningún otro envase por esa boca).

Si bien el supervisor esta dedicado en forma exclusiva a esta tarea, el mismo debe descansar sin hacer nada hasta que no se llene un cajón (es decir que no deberá estar chequeando constantemente si se llenó algún cajón).

La maquina deberá desplegar en una pantalla cual es el cajón que debe ser reemplazado, de forma que el supervisor no deba chequear todas las compuertas para identificar cual es el que se llenó.

Cuando una botella es ingresada a la boca se ejecuta automáticamente un V(x). X deberá ser definido de acuerdo a su solución

Se pide: Modelar este sistema utilizando semáforos, implementando el procedimiento tipo BOCA_ENTRADA y el procedimiento SUPERVISOR.

Nota: Se dispone de las funciones:

- **tipo_botella(boca)**: que da el tipo de botella que se encuentra en la boca
- **despliego_mensaje(mensaje)**: que despliega el cajon a cambiar en la pantalla
- **leer_mensaje()**: que lee de la pantalla que cajón hay que cambiar
- **cambiar_cajon (i)**: que cambia el cajon de tipo i
- **poner_botella (i,j)**: que pone la botella en el lugar j del cajon de tipo i.

Se deberá cuidar especialmente que 2 envases no sean puestos en el mismo lugar del cajón (que será considerado como un array de 6 posiciones).

==========

Solución:

```
int cant_botellas [1..5];
tipoSemaforo cajon [1..5];  // para cada cajón
tipoSemaforo boca [1..3];   // para cada boca
tipoSemaforo supervisor;    // para sincronizar al supervisor
tipoSemaforo mutex_pantalla // para mutuexcluir los accesos a la pantalla


procedure BOCA_ENTRADA (int num_boca)

int tipoBotella;

begin
   while true do
   begin
      P(boca[num_boca]);
      tipoBotella = tipo_botella(num_boca);
      P(cajon[tipoBotella]);
```

```
      cant_botellas[tipoBotella]++;
      poner_botella(tipoBotella, cant_botellas[tipoBotella]);
      if cant_botellas[tipoBotella] = 6 then
      begin
            P(mutex_pantalla);
            despliego_mensaje(tipoBotella);
            V(supervisor);
      end
      else
            V(cajon[tipoBotella]);
   end
end


procedure SUPERVISOR ()

int mensaje;

begin
   while true do
   begin
      P(supervisor);
      mensaje = leer_mensaje();
      V(mutex_pantalla);
      cambiar_cajon(mensaje);
      cant_botellas[mensaje] = 0;
      V(cajon[mensaje]);
   end
end


begin
   for i:= 1 to 5 do
      init(cajon[i], 1);
   for i:= 1 to 3 do
      init (boca[i], 0);
   init(supervisor, 0);
   init(mutex_pantalla, 1);
   for i:= 1 to 5 do
      cant_botellas[i] = 0;
   cobegin
      BOCA_ENTRADA(1);
      BOCA_ENTRADA(2);
      BOCA_ENTRADA(3);
      SUPERVISOR;
   coend
end.
================================================================================
```