Soluciones - Examen Enero de 2002

Cuestionario – Juego 1

1. Indique que función de las señaladas no realiza siempre la operación Open o Abrir archivo:
   a) Traducción de nombre Externo a Interno
   b) Creación de estructuras de acceso a memoria
   c) Asignación del área o extensión donde almacenar el archivo
   d) Verificación del derecho de acceso del mismo
2. Una Cache de memoria para un sistema multiprocesador debe:
   a) Amacenar lo último que se accedió
   b) Alacenar lo que tiene más probabilidad de volver a ser utilizado
   c) Amacenar solamente palabras de memoria con código y no datos
   d) Disponerse en las inmediaciones del control de memoria
3. Indique cual no es una ventaja resultado de que el sistema operativo tenga la posibilidad de realizar la asignación o alocación dinámica de sus tablas internas:
   a) Mayor eficiencia por menor overhead
   b) Evitar la necesidad de reconfigurar el sistema periódicamente
   c) Poder disponer de la cantidad de recursos adecuada a cada oportunidad
   d) Evitar límites máximos de la cantidad de semáforos disponibles
4. Indique que no es imprescindible para implementar una E/S no bloqueante en un Sistema Operativo
   a) El sistema debe incluir un DMA o equivalente
   b) El procesador debe contar con un esquema de interrupciones externas
   c) Instrucciones privilegiadas para implementar la E/S
   d) Un mecanismo que permita verificar el fin de la operación de E/S
5. Para un multiprocesador diseñado en torno a memoria común, indique cuál de las siguientes no es una ventaja derivadada del uso de procesadores con cache interna:
   a) Mayor velocidad de ejecución de los procesos
   b) Menor uso del bus de datos
   c) Acceso de escritura a la memoria más veloz
   d) Viabiliza el aumentar la cantidad de procesadores del sistema
6. En un sistema multiprocesador con memoria virtual por segmentación indique cual es la dificultad pincipal en oportunidad de reorganizar la memoria del mismo:
   a) La necesidad de evitar el acceso concurrente a un segnmento durante su desplazamiento
   b) Determinar que procesos tienen posibilidad de acceder al segmento
   c) Hallar el área libre dónde almacenarlo
   d) **A y B juntas**

7. En un sistema con paginación donde la memoria tiene un ciclo de acceso de 200ns y hay una probabilidad del 50% de hallar en el cache del procesador la entrada correspondiente en la tabla de traducción de páginas indique la velocidad promedio de acceso a memoria

   a) 200 ns

   b) 300 ns

   c) 400 ns

   d) Ninguna de los anterioeres

8. Dado el siguiente código:

```
Monitor RegionCritica;
      var sincRegion: condition;
      procedure entrar();
        begin
            sincRegion.wait();
        end;
      procedure salir();
        begin
            sincRegion.signal();
        end;
begin
      sincRegion.signal();
end;
```

   Si los clientes llaman a RegionCritica.entrar, acceden al recurso compartido y luego llaman a RegionCritica.salir.

   a) Resuelve el problema de mutuo exclusión (PME) pero tiene busy waiting.

   b) Dependiendo de la implementación del monitor puede llegar a resolver l PME.

   c) Tiene deadlock.

   d) Resuelve PME sin busy waiting.

   e) Ninguna de las anteriores,

9. El método del entrelazado permite identificar de manera clara:

   a) Deadlock.

   b) Posposición indefinida.

   c) La no resolución del problema de mutuo exclisión.

   d) Dos de las anteriores.

   e) Todas las anteriores.

10. El algoritmo del banquero se utiliza para determinar:

    a) La cantidad de recursos que está utilizando un determinado proceso.

    b) Si existe deadlock en el sistema.

    c) Si existen procesos huerfános en el sistema.

    d) Si el sistema está en un estado seguro.

    e) Determinar la utilización de CPU.

11. Cuándo un proceso ejecuta un V(s)
    a) Se bloquea sólo si s es -1.
    b) Se bloquea siempre.
    c) Se bloquea sólo si s es 0.
    d) Si no existe un proceso esperando, no hace nada.
    e) Ninguna de las anteriores.
12. ¿Cuál de las siguientes políticas se adecúa mejor a los sistemas de tiempo compartido?
    a) FIFO.
    b) LIFO.
    c) **Preemptivo**.
    d) Round robin.
    e) SJF.
13. A partir del estado pronto suspendido puedo pasar a:
    a) bloqueado suspendido.
    b) pronto activo.
    c) ejecutando.
    d) Todas las anteriores.
14. La fragmentación interna aparece:
    a) con las particiones variables.
    b) con las particiones fijas.
    c) con la segmentación.
    d) cuando utilizamos la política de asignación "best fit".
    e) Dos de las anteriores.
15. ¿Cuál de los siguientes algoritmos de planificación asegura que en todo momento ejecute aquel de mayor prioridad?
    a) FIFO.
    b) Round robin.
    c) Prioridad.
    d) **Prioridad "preemptive" o apropiativo**.
    e) Dos de los anteriores.
16. La construcción cobegin-coend:
    a) Proporciona un mecanísmo simple de sicronización.
    b) Permite generar concurrencia.
    c) Permite resolver cualquier grafo de precedencia.
    d) Dos de las anteriores.
    e) Todas las anteriores.

17. Un proceso hace referencia a las siguientes páginas: c a d b e b a b c d (en ese orden). La memoria pricipal aloja 4 marcos. Inicialmente en la memoria no hay cargada ninguna página.

    a) Si el sistema utiliza el algoritmo LRU se producen 5 fallos de página.

    b) Si el sistema utiliza el algoritmo LRU se producen 6 fallos de página.

    c) Si el sistema utiliza el algoritmo FIFO se producen 8 fallos de página.

    d) Si el sistema utiliza el algoritmo FIFO se producen 5 fallos de página.

    e) Dos de las anteriores.

    f) Ninguna de las anteriores.

18. Si se realizaron las siguientes solicitudes para un disco: 100 58 70 89 125 12 36 (en ese orden). La cabeza del disco se encuentra en el cilindro 96 moviendose hacia el cilindro 0.

    a) Si el sistema utiliza el algoritmo SCAN las solicitudes serán atendidas en este orden: 100 58 70 89 125 12 36.

    b) Si el sistema utiliza el algoritmo SCAN las solicitudes serán atendidas en este orden: 89 70 58 36 12 100 125.

    c) Si el sistema utiliza el algoritmo SCAN-C las solicitudes serán atendidas en este orden: 89 70 58 36 12 125 100.

    d) Si el sistema utiliza el algoritmo FIFO las solicitudes serán atendidas en este orden: 89 125 12 36 100 58 70.

    e) Dos de las anteriores.

19. En un sistema que utiliza gestión de memoria real con particiones dinámicas, en un momento determinado la lista de huecos está compuesta por huecos de tamaño 10 KB, 15 KB, 17 KB y 12 KB. ¿Qué política de asignación utiliza el sistema, si ante una solicitud de un bloque de 13 KB. elige el bloque de 15 KB?

    a) Peor ajuste.

    b) Mejor ajuste.

    c) Primer ajuste.

    d) Mejor o primer ajuste.

    e) Ninguna de las anteriores.

20. En cuanto a las operaciones "borrar archivo" y "cerrar archivo" sobre un archivo no vacío, es cierto que:

    a) Ambas eliminan sólo la información de ese archivo de la tabla de archivos abiertos del proceso que ha invocado la operación.

    b) Ambas eliminan la información de ese archivo, tanto de la tabla de archivos abiertos del proceso, como de la tabla general del sistema.

    c) Ambas no modifican el contenido del directorio que contiene a ese archivo.

    d) Ambas no tienen por qué modificar el contenido de los bloques de datos del archivo en disco.

21. Sea un sistema de memoria virtual paginada con páginas de 4kB. La memoria principal es de 64 MB. La Tabla de Páginas, que ocupa 2 MB, cuenta con descriptores de 2 bytes. ¿Cuál es el espacio de memoria virtual máximo disponible?

    a) 30 Mb.

    b) 64 MB.

    c) 4 GB.

    d) Un valor distinto de los anteriores.

22. La capacidad de almacenamiento de un mailbox:

    a) Determina si la comunicación será directa o indirecta.

    b) Determina si la comunicación será síncrona o asíncrona.

    c) No determina el tipo de comunicación a utilizar.

    d) Ninguna de las anteriores.

23. Si tenemos un sistema con paginado y 16 marcos, que está ejecutando un único proceso que ocupa 17 páginas, es cierto que:

    a) El algoritmo que mejor se comporta es el FIFO.

    b) El algoritmo que mejor se comporta es el LRU.

    c) El algoritmo que mejor se comporta es el de la 2ª oportunidad (o del reloj).

    d) Sin más datos, no puede saberse qué algoritmo de sustitución se comportará mejor

24. Una forma de reubicar estáticamente un archivo ejecutable consiste en sumar a todas las referencias relativas a memoria que se encuentren, la dirección de comienzo de memoria en donde se va a cargar dicho ejecutable. Para saber cuáles son estas referencias:

    a) El loader o cargador, genera una lista de las direcciones relativas que hacen referencia a memoria para que el linker pueda reubicarlas en el momento del linkedición.

    b) El linker genera dentro del archivo ejecutable una lista de direcciones a reubicar y el cargador, que conoce de antemano la estructura de todos los ejecutables, sabe dónde encontrar dicha lista.

    c) En el momento de la carga del programa en memoria, el compilador deja en la pila del proceso la lista de las direcciones a reubicar por el cargador.

    d) Es necesario contar con un registro de reubicación que contenga la dirección de memoria del comienzo de la lista de direcciones a reubicar.

25. En cuanto a procesos ligeros (threads) y pesados, es cierto que:

    a) En entornos monousuario no es posible el uso de proceso ligeros.

    b) Un proceso ligero es aquel que necesita poco tiempo de CPU para ejecutarse.

    c) Un proceso pesado es un proceso que tiene un alto grado de uso de espacio en disco.

    d) Varios procesos ligeros pueden compartir la misma tabla local de archivos abiertos.

26. Indicar cuál de las afirmaciones siguientes es cierta:

    a) El entorno de ejecución de un proceso ligero (thread) es mucho más reducido que el de un proceso tradicional ya que no incluye al del proceso.

    b) Un proceso ligero puede planificarse con las mismas políticas que los procesos tradicionales.

    c) Dos procesos ligeros pueden comunicarse pasándose información en la pila o "Stack".

    d) Dos procesos ligeros pueden comunicarse pasándose información en los registros del procesador.

27. Indicar cuál de las afirmaciones siguientes es cierta:

    a) **La comunicación entre dos procesos a través de un maibox de capacidad nula es siempre síncrona.**

    b) El envío de un mensaje por un mailbox de capacidad nula implica busy waiting hasta que el proceso receptor decida recibir.

    c) La comunicación entre dos procesos mediante el modelo de comunicación directa es siempre síncrona.

    d) Ninguna de las anteriores.

28. Las llamadas al Sistema...

    a) son el conjunto de servicios que ofrece el S.O. a nivel Hardware.

    b) son rutinas escritas en lenguaje de alto nivel que sirven de interfaz entre el S.O. y el Hardware.

    c) son servicios programados en código máquina que suministran la interfaz entre los drivers (manejadores) y el hardware.

    d) componen la interfaz de programación para acceder a los servicios del S.O.

29. Con Memoria Virtual, el tamaño de las páginas:

    a) Siempre debe ser múltiplo del tamaño de los marcos.

    b) Debe ser una potencia de 2 para disminuir la fragmentación interna.

    c) Debe ser una potencia de 2 para acelerar el proceso de traducción de direcciones.

    d) Cuanto menor sea su tamaño, se pierde menos tiempo en operaciones de E/S

30. ¿Quién se encarga de agrupar convenientemente en segmentos los objetos de un programa, para posteriormente poderlos referenciar mediante la tupla <segmento, desplazamiento>?

    a) El programador

    b) El compilador

    c) El cargador del sistema operativo

    d) La MMU (Unidad de Gestión de Memoria)

Cuestionario – Juego 2	Cuestionario – Juego 3	Cuestionario – Juego 4
1. c	1. d	1. b
2. d	2. d	2. d
3. d	3. e	3. b
4. e	4. e	4. d
5. c	5. d	5. b
6. b	6. d	6. a
7. b	7. c	7. d
8. d	8. b	8. c
9. d	9. d	9. b
10.e	10.b	10.c
11.e	11.d	11.b
12.d	12.b	12.a
13.d	13.a	13.c
14.c	14.d	14.c
15.b	15.c	15.d
16.d	16.b	16.b
17.b	17.c	17.c
18.d	18.b	18.d
19.b	19.a	19.d
20.a	20.c	20.e
21.d	21.c	21.c
22.c	22.d	22.b
23.b	23.b	23.b
24.c	24.c	24.d
25.b	25.d	25.d
26.a	26.d	26.e
27.c	27.e	27.e
28.c	28.c	28.d
29.d	29.b	29.d
30.b	30.b	30.c

Soluciones - Examen Enero de 2002

Ejercicio 1

En un ambiente académico se está desarrollando un prototipo para un sistema de ventanas.

Este prototipo está basado en un proceso que denominaremos "AdminVentanas", que se encarga de administrar el ambiente de ventanas. Cualquier proceso que quiera trabajar con ventanas, deberá conectarse a "AdminVentanas" y notificarle las operaciones realizadas.

Este proceso es el encargado de controlar en que orden se superponen las ventanas en la pantalla, que estado tiene actualmente cada ventana (abierta o icono), y notifica a los procedimientos cuando una región (rectángulo) de una ventana debe redibujarse porque ha sido expuesto (porque se abrió o porque otra ventana deja de estar encima de ella).

Cada proceso que utilice el sistema de ventanas, podrá invocar las siguientes funciones:

- `function conectarAdminVentanas():IndexConect;`

  Se conecta al proceso "AdminVentanas".

- `procedure desconectarAdminVentanas(ic:IndexConect);`

  Se desconecta del proceso "AdminVentanas".

- `function crearVentana(ic:IndexConect,rect:Rectangulo):IndexVentana;`

  Crea una ventana nueva, la función retorna el identificador de ventana único para las operaciones sobre esta. La ventana de muestra sobre todas las creadas anteriormente.

- `procedure destruirVentana(ic:IndexConect, iv:IndexVentana);`

  Destruye la ventana asociada al índice.

- `procedure abrirVentana(ic:IndexConect, iv:IndexVentana);`

  Abre la ventana asociada al índice (si estaba "iconificada", la despliega en la pantalla). Se respeta la posición anterior respecto a las demás ventanas.

- `procedure cerrarVentana(ic:IndexConect, iv:IndexVentana);`

  Cierra la ventana asociada al índice (si no estaba "iconificada", la "iconifica"). Se mantiene la posición respecto a las demás ventanas.

- `procedure al_frenteVentana(ic:IndexConect, iv:IndexVentana);`

  Se posiciona la ventana sobre todas las demás (se mantiene el estado).

- `procedure al_fondoVentana(ic:IndexConect, iv:IndexVentana);`

  Se posiciona la ventana tras todas las demás (se mantiene el estado).

Además, el proceso deberá aceptar del administrador de ventanas el siguiente evento:

- `<`**`REDIBUJAR`**`, IndexVentana, Rectangulo>`

  Le informa al proceso que debe redibujar la zona definida por `Rectángulo` en la ventana `IndexVentana`.

Se pide implementar las funciones y procedimientos anteriores y la inicialización del sistema de forma de mantener el sistema de ventanas funcionando usando mailboxes y semaforos (deberá especificarse la semántica seleccionada). Por lo tanto deberá definirse el código que ejecuta cada proceso y el código asociado del lado del administrador de ventanas.

Observación 1

Se considerarán definidas las siguientes operaciones sobre las siguientes estructuras:

```
type ModoVentana = [ABIERTO, CERRADO];

struct nodoListaVentanas =
     ic:IndexConect,
     iv:IndexVentana,
     rect:Rectangulo,
     modo:ModoVentana
end struct;
type ListaVentanas List of nodoListaVentanas;

struct nodoListaRedibujarVentana =
     ic:IndexConect,
     iv:IndexVentana,
     rect:Rectangulo,
end struct;
type ListaRedibujarVentana List of nodoListaRedibujarVentana;
```

- `function InitLV():ListaVentanas;`
- ```
  procedure AgergarLV(var lv:ListaVentanas,
                      ic:IndexConect,
                      iv:IndexVentana,
                      rect:Rectangulo);
  ```
  Agrega el nuevo nodo al inicio (al frente) de la lista y modo `ABIERTO`.
- ```
  procedure BorrarLV(var lv:ListaVentanas,
                     iv:IndexVentana);
  ```
- ```
  procedure GetLV(lv:ListaVentanas,
                  iv:IndexVentana,
                  var idx:integer,
                  var rect:Rectangulo,
                  var modo:ModoVentana);
  ```
  Selector sobre la lista a través de `iv`, donde `idx` indica en que posición de la lista se encuantra el nodo.
- ```
  procedure CambioModo(var lv:ListaVentanas,
                       iv:IndexVentana,
                       modo:ModoVentana);
  ```

- `procedure AlFrente(var lv:ListaVentanas;`
  `iv:IndexVentana);`

  Coloca el nodo que contiene `iv` al inicio (al frente) de la lista.

- `procedure AlFondo(var lv:ListaVentanas;`
  `iv:IndexVentana);`

  Coloca el nodo que contiene `iv` al final (al fondo) de la lista.

- `function NotiDibujarLV(var lv:ListaVentanas;`
  `idx:integer;`
  `rect:Rectángulo): ListaRedibujarVentana;`

  Retorna la lista de regiones a redibujar si se descubre la región definida por `rect` a partir de la posición `idx` en `lv`.

- `function PrimeroLRV(lrv: ListaRedibujarVentana)`
  `: nodoListaRedibujarVentana;`

  Retorna el primer elemento de la lista `lrv`.

- `function RestoLRV(lrv: ListaRedibujarVentana)`
  `: ListaRedibujarVentana;`

  Retorna el primer resto de la lista `lrv`.

- `function VaciaLRV(lrv: ListaRedibujarVentana): Boolean;`

  Retorna si la lista `lrv` es vacía.

Observación 2

Se considera que la máxima cantidad de procesos en el sistema es la constante `MAX_PROC` (`IndexConect` está ente `1` y `MAX_PROC`).

Además se considera que esa constante es pequeña, de modo que es admisible definirse arreglos de tamaño `MAX_PROC` para cualquier tipo de datos.

Observación 3

Se considera que la cantidad de ventanas creada en el sistema desde que se levanta hasta que se baja nunca superará el rango del tipo integer (por lo tanto `IndexVentana` se define como `integer`).

```
=========================================================================

Usaremos mailboxes infinitos.
Definimos las siguientes estructuras adicionales:

*************************************************************************

type IndexConect = 1 .. MAX_PROC;
type IndexVentana = integer;

struct Notificar =
   op: [INDICE_VENT, REDIBUJAR],
   iv:IndexVentana,
   rect:Rectangulo
end struct;

type Conexiones Array[IndexConect] of mailbox of Notificar;

struct OperVentanas =
   op: [CREATE_VENTANA, DELETE_VENTANA, OPEN_VENTANA,
        CLOSE_VENTANA, FRONT_VENTANA, BACK_VENTANA],
   vent:IndexVentana,
   rect:Rectangulo,
   conect: IndexConect
end struct;

var conect: Conexiones;
var av_mbx: Mailbox of OperVentanas;
var ult_idx_vent: IndexVentana;

var canal_mbx: Mailbox of IndexConect;

-------------------------------------------------------------------------
function conectarAdminVentanas():IndexConect
var nueva_pos: IndexConect;
begin
   receive(canal_mbx, nueva_pos);
   return(nueva_pos);
end function conectarAdminVentanas;

procedure desconectarAdminVentanas(ic:IndexConect);
begin
   send(canal_mbx, ic);
end procedure desconectarAdminVentanas;

function crearVentana(ic:IndexConect,rect:Rectangulo):IndexVentana;
var oper:OperVentanas;
    noti: Notificar;
begin
   oper.operacion := CREATE_VENTANA;
   oper.conect := ic;
   oper.rect := rect;
   send(av_mbx, oper);
   receive(conect[ic], noti);
   return(noti.iv);
end procedure crearVentana;
```

```
procedure destruirVentana(ic:IndexConect, iv:IndexVentana);
var oper:OperVentanas;
begin
   oper.operacion := DELETE_VENTANA;
   oper.vent := iv;
   send(av_mbx, oper);
end procedure destruirVentana;

procedure abrirVentana(ic:IndexConect, iv:IndexVentana);
var oper:OperVentanas;
begin
   oper.operacion := OPEN_VENTANA;
   oper.vent := iv;
   send(av_mbx, oper);
end procedure abrirVentana;

procedure cerrarVentana(ic:IndexConect, iv:IndexVentana);
var oper:OperVentanas;
begin
   oper.operacion := CLOSE_VENTANA;
   oper.vent := iv;
   send(av_mbx, oper);
end procedure cerrarVentana;

procedure al_frenteVentana(ic:IndexConect, iv:IndexVentana);
var oper:OperVentanas;
begin
   oper.operacion := FRONT_VENTANA;
   oper.vent := iv;
   send(av_mbx, oper);
end procedure al_frenteVentana;

procedure al_fondoVentana(ic:IndexConect, iv:IndexVentana);
var oper:OperVentanas;
begin
   oper.operacion := BACK_VENTANA;
   oper.vent := iv;
   send(av_mbx, oper);
end procedure al_ fondoVentana;
-------------------------------------------------------------------------
procedure Notificar(list_vent: ListaVentanas,
                    rect:Rectangulo, idx:integer)
var
   noti:Notificar;
   lrv:ListaRedibujarVentana;
   nlrv:nodoListaRedibujarVentana;
begin
   lrv := NotiDibujarLV(list_vent, idx, rect);
   noti.op := REDIBUJAR;
   while (not VaciaLRV(lrv)) do
      begin
         nlrv := PrimeroLRV(lrv);
         noti.iv := nlrv.iv;
         noti.rect := nlrv.rect;
         send(conect[nlrv.ic], noti);
         lrv := RestoLRV(lrv);
   end while
end procedure Notificar;
```

```
procedure AdminVentanas();
var list_vent:ListaVentanas;
    oper:OperVentanas;
    noti:Notificar;
    idx:integer;
    rect:Rectangulo;
    modo:ModoVentana;
    idx_ic:IndexConect;
begin
   list_vent := InitLV();

   for idx_ic = 1 to MAX_PROC do
      send(canal_mbx, idx_ic);
   end for

   while(true) do
   begin
      receive(av_mbx, oper);
      case of oper.operacion
          CREATE_VENTANA:
              begin
                 ult_idx_vent : = (ult_idx_vent + 1);
                 noti.op := INDICE_VENT;
                 noti.iv := ult_idx_vent;
                 send(conect[oper.ic], noti);
                 AgergarLV(list_vent, oper.conect, ult_idx_vent,
                           oper.rect);
                 // Notifico para que se dibuje la nueva ventana.
                 noti.op := REDIBUJAR;
                 noti.iv := ult_idx_vent;
                 noti.rect := oper.rect;
                 send(conect[oper.ic], noti);
              end
          DELETE_VENTANA:
              begin
                 GetLV(list_vent, oper.iv, idx, rect, modo);
                 BorrarLV(list_vent, oper.iv);
                 if (modo = ABIERTO) then
                    // Notifico para que se dibujen la ventanas que
                    // están detras de la eliminada y que quedaron
                    // descubiertas.
                    Notificar(list_vent, rect, idx);
                 end if;
              end
```

```
            OPEN_VENTANA:
                begin
                   GetLV(list_vent, oper.iv, idx, rect, modo);
                   if (modo = CERRADO) then
                      CambioModo(list_vent, oper.vent, ABIERTO);
                     // Notifico para que se dibuje la ventana abierta.
                      noti.op := REDIBUJAR;
                      noti.iv := oper.vent;
                      noti.rect := oper.rect;
                      send(conect[oper.ic].mbx, noti);
                   end if;
                end
            CLOSE_VENTANA:
                begin
                   GetLV(list_vent, oper.vent, idx, rect, modo);
                   if (modo = ABIERTO) then
                      CambioModo(list_vent, oper.vent, CERRADO);
                      // Notifico para que se dibujen la ventanas que
                      // están detras de la eliminada y que quedaron
                      // descubiertas.
                      Notificar(list_vent, rect, idx);
                   end if;
                end
            FRONT_VENTANA:
                begin
                   AlFrente(list_vent, oper.vent);
                   GetLV(list_vent, oper.vent, idx, rect, modo);
                   if (modo = ABIERTO) then
                      // Notifico para que se dibuje la ventana al
                      // frente.
                      noti.op := REDIBUJAR;
                      noti.iv := oper.vent;
                      noti.rect := rect;
                      send(conect[oper.ic], noti);
                   end if;
                end
            BACK_VENTANA:
                begin
                   GetLV(list_vent, oper.vent, idx, rect, modo);
                   AlFondo(list_vent, oper.vent);
                   if (modo = ABIERTO) then
                      // Notifico para que se dibujen la ventanas que
                      // estaban detras de la enviada al fondo y que
                      // quedaron descubiertas.
                      Notificar(list_vent, rect, idx);
                   end if;
                end
         end case;
      end while;
   end procedure AdminVentanas;
```

==========

Ejercicio 2

Se considera un sistema de preparación de cócteles de acuerdo al estado etílico del cliente denominado Coctelera-Pro-Salud 2002 (CPS2002). Su estructura y funcionamiento es el siguiente:

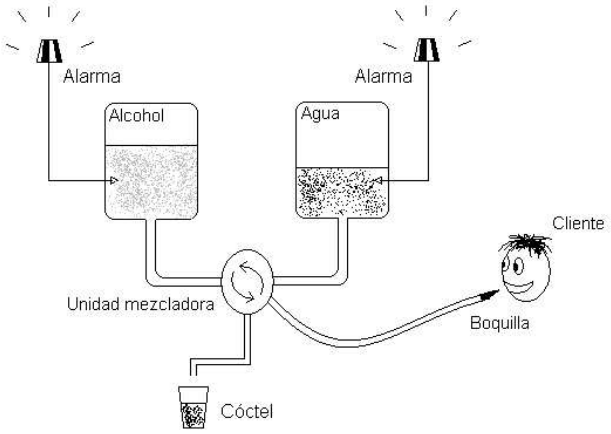
Se dispone de dos tanques (uno para una bebida alcohólica y otro para agua), un dispositivo que realiza un test de alcoholemia (nivel de alcohol en la sangre) y un botón de solicitud. Cuando el cliente desea una bebida, debe presionar el botón de solicitud. A partir de ese momento la maquina espera un tiempo máximo de 10s a que el cliente realice el test de alcoholemia (soplando por la bombilla) y luego prepara el cóctel teniendo en cuenta el resultado del test. Si pasados los 10s el test no se completó se cancela la operación.

El cóctel siempre es de 100cc, combinando bebida alcohólica y agua según la expresión:

(100 - X)cc bebida alcohólica + Xcc agua

donde X es el resultado del test de alcoholemia. O sea: la máquina rebaja las bebidas a medida que el cliente está cada vez más borracho.

Si el contenido de alguno de los tanques disminuye demasiado deberá sonar una alarma asociada al tanque que se está agotando. Acto seguido, la mezcladora debera aguardar a que los tanques sean recargados para continuar con el proceso de preparacion de la bebida.



CPS2002 – Detalle de funcionamiento

Se pide:

1. Implementar utilizando ADA las tareas que componen la CPS2002.
2. Implementar utilizando las primitivas Send/Receive las tareas que componen la CPS2002. Suponga que dispone de un proceso timer, con **pid = TIMER_PID**, el cual recibe en un mensaje la cantidad de segundos a esperar, y le avisa al proceso que solicito el pedido cuando este se cumplio. Suponga despreciable el tiempo de envio o recepción de los mensajes. Defina el formato de los mensajes a intercambiar con este proceso según sus necesidades.

Notas:

- En ambos casos, las tareas deberán implementarse de la forma mas eficiente posible.
- Las válvulas de agua y bebida alcohólica dejan pasar bebida a razón de 1cc cada 100ms.
- El resultado de la prueba de alcoholemia puede obtenerse invocando la funcion **controlarBoquilla()**, la cual devuelve –1 si el test no pudo realizarse o un numero entre 0 y 100 para indicar el % de alcohol en la sangre. Esta funcion demorara un maximo de 10 segundos en retornar un valor desde la primer invocación a la misma.

- Se dispone de las funciones **abrirLlavePaso(tpoliq:Integer)** y **cerrarLlavePaso(tpoliq:Integer)** las cual abren y cierran las llaves de los tanques.
- Se dispone de las funciones **encenderAlarma(tpoliq:Integer)** y **apagarAlarma(tpoliq:Integer)** las cual prenden y apagan la alarma de un tanque
- Se dispone de las funciones **nivelLiquido(tpoliq: Integer)** la cual devuelve el nivel de liquido de un tanque.
- En todos los casos, las funciones auxiliares reciben como parámetro un número entero que indica el tipo de liquido (o tanque) sobre el que actuaran. 0 para agua, 1 para alcohol.

```
================================================================================
1)
program CPS2002 is;
var
      tqueA, tqueH      : Tanque;
      mezclar           : Mezcladora;
      testear           : Testeador;

{ Tanque, modulo general para el manejo de tanques con alarma }
task Tanque is
var
      nivel, cntPasar:  Integer;
      tpoliq:           Integer;

begin
      loop
            { Abre la llave de paso (si corresponde) }
            accept Abrir(cntPasarIn: Integer; tpoliqIn: Integer) {
                  cntPasar = cntPasarIn;  { Cantidad en CC que debe pasar }
                  tpoLiq   = tpoLiqIn;    { Tipo de liquido de este tanque }
            end

            nivel = nivelLiquido(tpoLiq);

            if cntPasar > nivel and cntPasar > 0 then
                  prenderAlarma(tpoLiq);
                  { Punto de encuentro que ejecuta la tarea rellenadora. No se
                    pide implementacion de esta. }
                  accept Rellenado();

                  apagarAlarma(tpoLiq);
                  nivel = nivelLiquido(tpoLiq);
            end

            if cntPasar > 0 then
                  abrirLlavePaso(tpoLiq);
                  delay (cntPasar/100);   { Milisegundos a esperar p/todo el
                                            liquido }
                  cerrarLlavePaso(tpoLiq);
                  mezclar.Finalizado();
            end;
      endloop;
end { Tanque }

{ Tarea que representa a la unidad mezcladora. Es la encargada de coordinar la p
  preparacion de las bebidas }
task Mezcladora is
var
      pjealco: Integer;       { % de alcohol en la sangre }
begin
      loop
            accept BotonOprimido() {}; { Cuando aprieten el boton, arrancamos }
            pjealco = -1;
            testear.iniciar();
            select
                  accept TesteoRealizado(pjeIn: Integer) begin
                        pjealco = pjeIn;
                  end;
            or
                  delay 10
```

```
            end
            if pjealco <> -1 then { Testeo exitoso! }
                  tqueH.Abrir(100 - pjealco, 0);
                  tqueA.Abrir(pjealco, 1);
                  accept Finalizado();    { Testeo exitoso! }
                  accept Finalizado();    { Testeo exitoso! }{ Testeo exitoso! }
            end
      endloop
End { Mezcladora }

task Testeador is
var
      pjealco: Integer;       { % de alcohol en la sangre }
begin
      accept Iniciar();
      { pooleamos sobre la boquila hasta que devuelva resultado }
      pjealco := -1;
      while pjealco == -1 do begin
            pjealco = controlarBoquilla();
      end
      { Se asume que esta tarea no ejecutara el encuentro pasados los 10s, desde
        la primer invocacion }
      mezclar.TesteoRealizado();
end { Testeador }

{ Inicializacion (Tareas globales) }
begin
      tqueA   = new Tanque();
      tqueH   = new Tanque();
      mezclar = new Mezcladora();
      testear = new Testeador();
end
```

2)

Send: Asincronico con nombrado explicito
Receive: Sincronico con nombrado implicito.

Formato de los mensajes:

```
TipoMensajeTiempo ::= record
      origen: String;
      tiempo: Integer;
end;
TipoMensajeTest ::= record
      pjealco: Integer;
end;
TipoMensajeTanque ::= record
      tipoLiq, cntPasar: Integer;
end;


program CPS2002;

{ Tanque de agua }
procedure TanqueA();

var
      nivel: Integer;
      msg:        TipoMensajeTanque;
      msgtimer:   TipoMensajeTiempo;

begin
      while true do begin
            { Abre la llave de paso (si corresponde) }
            receive(msg);

            nivel = nivelLiquido(msg.tipoLiq);
            if msg.cntPasar > nivel and msg.cntPasar > 0 then
                  prenderAlarma(msg.tipoLiq);
                  { Punto de encuentro que ejecuta la tarea rellenadora. No se
                    pide implementacion de esta. }
                  receive(null);

                  apagarAlarma(msg.tipoLiq);
                  nivel = nivelLiquido(msg.tipoLiq);
            end

            if cntPasar > 0 then
                  abrirLlavePaso(msg.tipoLiq);

                  { Milisegundos a esperar p/todo el liquido }
                  msgtimer.tiempo = msg.cntPasar/100;
                  send("TIMER_PID", msgtimer);
                  receive(null)

                  cerrarLlavePaso(msg.tipoLiq);
                  send("Mezcladora", null);
            end
      endwhile
end;
```

```
{ Tanque de alcohol, idem tanque de agua }
procedure TanqueH();
……

{ Tarea que representa a la unidad mezcladora. Es la encargada de coordinar la p
  preparacion de las bebidas }
procedure Mezcladora();
var
      pjealco:    Integer;    { % de alcohol en la sangre }
      msgtque:    TipoMensajeTanque;
      msgtest:    TipoMensajeTest;
      msgtimer:   TipoMensajeTiempo;

begin
      loop
            { Aviso del boton para inciar el proceso.
              Ignora mensajes llegados fuera de tiempo del TIMER }
            while not(arrancar) do
                  receive(msg);
                  arrancar = (msg.origen == "BOTON");
            endloop

            pjealco = -1;
            send("Testeadora",null);

            { Milisegundos a esperar p/todo el liquido }
            msgtimer.tiempo = 10;
            send("TIMER_PID", msgtimer);

            receive(msg);
            { Testeo exitoso, sino se inicia preparacion de un nuevo
              trago }
            if msg.orig = "Testeadora" and msg.pjealco <> -1 then
                  msgtque.tipoLiq  = 0;
                  msgtque.cntPasar = 100 - pjealco;
                  send("TanqueH",msgtque);

                  msgtque.tipoLiq  = 1;
                  msgtque.cntPasar = pjealco;
                  send("TanqueA",msgtque);

                  receive(null);
                  receive(null);
            end
      endloop
End { Mezcladora }

procedure Testeador();
var
      pjealco: Integer;       { % de alcohol en la sangre }
begin
      receive(null);
      { pooleamos sobre la boquila hasta que devuelva resultado }
      pjealco := -1;
      while pjealco == -1 do begin
            pjealco = controlarBoquilla();
      end
      msg.pjealco = pjealco;
      send("Mezcladora",msg);
end { Testeador }
```

```
begin
      cobegin
            Testeador();
            Mezcladora();
            TanqueA();
            TanqueH();
      coend;
end;
```

==========