Práctica de Repaso Memoria Distribuida

Pasaje de mensajes

1. En una oficina existen 100 empleados que envían documentos para imprimir en 5 impresoras compartidas. Los pedidos de impresión son procesados por orden de llegada y se asignan a la primera impresora que se encuentre libre:
   a) Implemente un programa que permita resolver el problema anterior usando PMA.

```
chan imprimir(text);
chan pedido();

procedure Empleado[id: 0.. 99]{
    text doc;
    doc= generarDocumento();
    send imprimir (doc);
    send pedido();
}

procedure Impresora [id: 0.. 4]{
    text doc;
    while (true){
        receive pedido();
        receive imprimir (doc);
        imprimir (doc);
    }

}
```

   b) Resuelva el mismo problema anterior pero ahora usando PMS.

```
procedure Empleado[id: 0.. 99]{
    text doc;
    doc= generarDocumento();
    Administrador!imprimir (doc);
}

procedure Administrador (){
    cola documentos;
    cola iLibres;
    do Empleado[*]?imprimir(doc) -> {
        if !iLibres.vacia()
            Impresora[iLibres.pop()]!imprimir(doc);
            else
                documentos.push(doc);
        }
    [] Impresora[*]?libre(idImpresora) -> {
        if !documentos.vacia()
            Impresora[idImpresora]!imprimir(documentos.pop())
        else
            iLibres.push(idImpresora);
        }
    od
}

}

procedure Impresora [id: 0.. 4]{
    text doc;
    while (true){
        Administrador!libre(id);
        Administrador?imprimir(doc);
        imprimir (doc);
    }
}
```

```
}
```

2. Resolver el siguiente problema con PMS. En la estación de trenes hay una terminal de SUBE que debe ser usada por P personas de acuerdo con el orden de llegada. Cuando la persona accede a la terminal, la usa y luego se retira para dejar al siguiente. Nota: cada Persona usa sólo una vez la terminal.

```
process Persona [id: 0.. P-1]{
    Terminal!solicitarUso(id);
    Terminal?usar();
    //usa la terminal
    Terminal!dejarTerminal();
}

process Terminal() {
    int idPersona;
    for i:= 1 to P do
    begin
        Persona[*]?solicitarUso(idPersona);
        Persona[idPersona]!usar();
        Persona[idPersona]?dejarTerminal();
    end;
}

-----------------------

    process Persona [id: 0.. P-1]{
    Terminal!solicitarUso(id);
    Terminal?usar();
    //usa la terminal
    Terminal!dejarTerminal();
}

process Terminal(){
```

```
    cola espera;
    boolean libre = true;
    int id

    do Persona[*]?solicitarUso(id) ->{
        if (!libre)
            espera.push(id);
        else{
            Persona[id]!usar();
            libre:= false
        }
        }
    [] Persona[*]?dejarTerminal() -> {
        if (espera.vacia())
            libre:= true;
        else
            Persona[espera.pop()]!usar();
        }
    od
}
```

3. Resolver el siguiente problema con PMA. En un negocio de cobros digitales hay P personas que deben pasar por la única caja de cobros para realizar el pago de sus boletas. Las personas son atendidas de acuerdo con el orden de llegada, teniendo prioridad aquellos que deben pagar menos de 5 boletas de los que pagan más. Adicionalmente, las personas embarazadas tienen prioridad sobre los dos casos anteriores. Las personas entregan sus boletas al cajero y el dinero de pago; el cajero les devuelve el vuelto y los recibos de pago

```
chan menos5Boletas(int, cola, real);
chan mas5Boletas(int, cola, real);
chan embarazadas(int, cola, real);
chan finalizarPago[P](real, cola);
chan pedido();
```

```
process Persona[id: 0.. P-1]{
    cola boletas= generarBoletas();
    real dineroPago;
    cola recibos;
    boolean embarazada = //indica si está embarazada
    if (embarazada)
        send embarazadas(id, boletas, dineroPago)
    else
        if (boletas.size() < 5 )
            send menos5Boletas(id, boletas, dineroPago)
        else
            send mas5Boletas(id, boletas, dineroPago)
    send pedido();
    receive finalizarPago[id](vuelto, recibos);
}

process Cajero(){
    int idPersona;
    cola boletas;
    real vuelto;
    cola recibos;
    real dineroPago;
    while (true){
        receive pedido();
        if (!empty embarazadas)
            receive embarazadas(idP, boletas, dineroPago);
        else{
            if (!empty menos5Boletas)
                receive menos5Boletas(idP, boletas, dineroPag
            else
                receive mas5Boletas(idP, boletas, dineroPago)
        }
        vuelto= generarVuelto(dineroPago, boletas);
        recibos= generarRecibos(boletas);
        send finalizarPago[idP](vuelto, recibos);
    }
}
```

ADA

1. Resolver el siguiente problema. La página web del Banco Central exhibe las diferentes cotizaciones del dólar oficial de 20 bancos del país, tanto para la compra como para la venta. Existe una tarea programada que se ocupa de actualizar la página en forma periódica y para ello consulta la cotización de cada uno de los 20 bancos. Cada banco dispone de una API, cuya única función es procesar las solicitudes de aplicaciones externas. La tarea programada consulta de a una API por vez, esperando a lo sumo 5 segundos por su respuesta. Si pasado ese tiempo no respondió, entonces se mostrará
   vacía la información de ese banco.

```
Procedure BancoCentral is
    TASK TYPE API is
        ENTRY devolverCotización(venta: OUT real; compra: OUT
    end API

    TASK Actualizador;

    apis= array (1..20) of API

    TASK BODY ACTUALIZADOR is
        cotizaciones: cola;
        venta, compra: real;
    BEGIN
        loop
        for i:= 1 to 20 do
        BEGIN
            SELECT
                apis[i].devolverCotización(venta, compra);
```

```
                cotizaciones.push(venta, compra);
                or delay(5)
                    cotizaciones.push(' ', ' ');
            END SELECT
        END

      for i:= 1 to 20 do
          print (cotizaciones.pop());
         //espera un periodo de tiempo
     end loop
    END ACTUALIZADOR

    TASK BODY API is
        valorVenta, valorCompra: real;
        BEGIN
            loop
            valorVenta:=ventaActual();
            valorCompra:=compraActual();
            accept devolverCotización(venta: out real; compra
                venta:= valorVenta;
                compra: valorCompra;
            end devolverCotización
            end loop
        END API
BEGIN
    null
END;
```

2. Resolver el siguiente problema. En un negocio de cobros digitales hay P personas que deben pasar por la única caja de cobros para realizar el pago de sus boletas. Las personas son atendidas de acuerdo con el orden de llegada, teniendo prioridad aquellos que deben pagar menos de 5 boletas de los que pagan más. Adicionalmente, las personas ancianas tienen prioridad sobre los dos casos anteriores. Las personas entregan sus boletas al cajero y el dinero de pago; el cajero les devuelve el vuelto y los recibos de pago.

```
Procedure Negocio is

    TASK TYPE Persona;

    TASK Cajero is
        ENTRY colaPersonasMenos5Boletas (boletas: in cola;
        ENTRY colaPersonasMas5Boletas (boletas: in cola; v
        ENTRY colaPersonasAncianas  (boletas: in cola; vue
    end Cajero

    personas= array (1..P) of Persona;

    TASK BODY Persona is
        boletas: cola;
        anciana: boolean;
        vuelto: real;
        recibo: text;
    BEGIN
     boletas:= cargarMisBoletas();
     if (anciana)
            Cajero.colaPersonasAncianas(boletas, vuelto, r
    else
        BEGIN
         if (boletas.size() < 5)
             Cajero.colaPersonasMenos5Boletas (boletas, vu
         else
             Cajero.colaPersonasMas5Boletas (boletas, vuel
        END
 END Persona

 TASK Cajero is

 BEGIN
     for i:= 1 to p loop
         SELECT
             accept colaPersonasAncianas (boletas: in cola
                 //procesa las boletas;
                 vuelto:= obtenerVuelto(boletas);
```

```
                recibo:= obtenerRecibo(boletas);
               end colaPersonasAncianas;
           or
               WHEN (colaPersonasAncianas'count = 0)
                    accept colaPersonasMenos5Boletas (bol
                //procesa las boletas;
                vuelto:= obtenerVuelto(boletas);
                recibo:= obtenerRecibo(boletas);
               end c;olaPersonasMenos5Boletas;
           or
               WHEN (colaPersonasAncianas'count = 0 and c
                accept colaPersonasMas5Boletas (boletas:
                //procesa las boletas;
                vuelto:= obtenerVuelto(boletas);
                recibo:= obtenerRecibo(boletas);
               end colaPersonasMasBoletas;
           END SELECT
        end loop
    END Cajero

BEGIN
     null;
END
```

3. Resolver el siguiente problema. La oficina central de una empresa de venta de indumentaria debe calcular cuántas veces fue vendido cada uno de los artículos de su catálogo. La empresa se compone de 100 sucursales y cada una de ellas maneja su propia base de datos de ventas. La oficina central cuenta con una herramienta que funciona de la siguiente manera: ante la consulta realizada para un artículo determinado, la herramienta envía el identificador del artículo a las sucursales, para que cada una calcule cuántas veces fue vendido en ella. Al final del procesamiento, la herramienta debe conocer cuántas veces fue vendido en total, considerando todas las sucursales. Cuando ha terminado de procesar un artículo comienza con el siguiente (suponga que la herramienta tiene una función generarArtículo() que retorna el siguiente ID a consultar). Nota: maximizar la concurrencia. Existe una función ObtenerVentas(ID) que

retorna la cantidad de veces que fue vendido el artículo con identificador ID en la base de la sucursal que la llama

```
Procedure Oficina is

    TASK TYPE Sucursal is
        ENTRY obtenerArticuloAProcesar(id: in int)
        ENTRY devolverCantidadDeVentas(cant: out int);
    END Sucursal;

    TASK Herramienta;


    sucursales = array (1..100) of Sucursal

    TASK BODY Herramienta is
        idArticulo: int;
        cantidadPorArticulo: int;
        cantPorSucursal:int
    BEGIN
        loop
            cantidadPorArticulo:= 0;
            idArticulo:=generarArticulo();

            for i:= 1 to 100 do
                sucursales(i).obtenerArticuloAProcesar(idArti

            for i:= 1 to 100 do
            begin
                sucursales(i).devolverCantidadDeVentas(cantPo
                cantidadPorArticulo+= cantPorSucursal
            end
            print('Total de ventas del articulo ', idArticulo
        end loop
    End Herramienta

    TASK BODY Sucursal is
        cantidadArticulo: int
```

```
        idA: int
    BEGIN
        loop
            accept obtenerArticuloAProcesar(idArticulo: in in
                idA:= idArticulo;
            end obtenerArticuloAProcesar;
            cantidadArticulo:= obtenerVentas(idA);
            accept devolverCantidadDeVentas(cant: out int) do
                cant:= cantidadArticulo;
            end devolverCantidadDeVentas;
        end loop
    END Sucursal
BEGIN
    null
END

-----------------------------------------------------------------

Procedure Oficina is

    TASK TYPE Sucursal


    TASK Herramienta;
        ENTRY obtenerArticuloAProcesar(id: out int)
        ENTRY devolverCantidadDeVentas(cant: in int);
    END Herramienta

    sucursales = array (1..100) of Sucursal

    TASK BODY Herramienta is
        idArticulo: int;
        cantidadPorArticulo: int;
        cantPorSucursal:int
    BEGIN
        loop
            cantidadPorArticulo:= 0;
            idArticulo:=generarArticulo();
```

```
            for i:= 1 to 200 do
            begin
                select
                    accept obtenerArticuloAProcesar(id: out i
                        id:= idArticulo
                    end obtenerArticuloAProcesar
                    or
                        accept devolverCantidadDeVentas(cant:
                            cantidadPorArticulo+= cant
                        end devolverCantidadDeVentas
                end select
            end
            print('Total de ventas del articulo ', idArticulo
        end loop
    End Herramienta

    TASK BODY Sucursal is
        cantidadArticulo: int
        idA: int
    BEGIN
        loop
            Herramienta.obtenerArticuloAProcesar(idA)
            cantidadArticulo:= obtenerVentas(idA);
            accept devolverCantidadDeVentas(cantidadArticulo)
        end loop
    END Sucursal
BEGIN
    null
END
```