Práctica 5

1. Se requiere modelar un puente de un único sentido que soporta hasta 5 unidades de peso. El peso de los vehículos depende del tipo: cada auto pesa 1 unidad, cada camioneta pesa 2 unidades y cada camión 3 unidades. Suponga que hay una cantidad innumerable de vehículos (A autos, B camionetas y C camiones). Analice el problema y defina qué tareas, recursos y sincronizaciones serán necesarios/convenientes para resolverlo.
   a. Realice la solución suponiendo que todos los vehículos tienen la misma prioridad.

   ```

   ```

   b. Modifique la solución para que tengan mayor prioridad los camiones que el resto de los vehículos.

   ```

   ```

2. Se quiere modelar el funcionamiento de un banco, al cual llegan clientes que deben realizar un pago y retirar un comprobante. Existe un único empleado en el banco, el cual atiende de acuerdo con el orden de llegada.
   a) Implemente una solución donde los clientes llegan y se retiran sólo después de haber sido
   atendidos.

   ```
   Procedure Banco is

       TASK TYPE Cliente;

       TASK Empleado is
           ENTRY realizarPago(pago: in real; comprobante: out
       End Empleado

       clientes = array (1..C) of Cliente;
   ```

```
    TASK BODY Cliente is
        pago: real;
        comprobante: text;
        BEGIN
            pago:= retornarMontoAPagar();
            Empleado.realizarPago(pago, comprobante);
        END Cliente

    TASK BODY Empleado is
        pago:real;
        BEGIN
            loop
                accept realizarPago (pago: in real; compro
                    comprobante:= cobrar(pago);
                end realizarPago;
            end loop;
        END Empleado;

        BEGIN
            null
        END Banco;
```

b) Implemente una solución donde los clientes se retiran si esperan más de 10 minutos para
realizar el pago.

```
Procedure Banco is

    TASK TYPE Cliente;

    TASK Empleado is
        ENTRY realizarPago(pago: in real; comprobante: out
    End Empleado
```

```
    clientes = array (1..C) of Cliente;

    TASK BODY Cliente is
        pago: real;
        comprobante: text;
        BEGIN
            pago:= retornarMontoAPagar();
            SELECT
                Empleado.realizarPago(pago, comprobante);
            or delay (10)
                //se retira si pasan 10 minutos y no fue a
            END SELECT
        END Cliente

    TASK BODY Empleado is
        pago:real;
        BEGIN
            loop
                accept realizarPago (pago: in real; compro
                    comprobante:= cobrar(pago);
                end realizarPago;
            end loop;
        END Empleado;

        BEGIN
            null
        END Banco;
```

c) Implemente una solución donde los clientes se retiran si no son atendidos
inmediatamente.

```
Procedure Banco is

    TASK TYPE Cliente;
```

```
TASK Empleado is
    ENTRY realizarPago(pago: in real; comprobante: out
End Empleado

clientes = array (1..C) of Cliente;

TASK BODY Cliente is
    pago: real;
    comprobante: text;
    BEGIN
        pago:= retornarMontoAPagar();
        SELECT
            Empleado.realizarPago(pago, comprobante);
        else
            //se retira si no es atendido inmediatamen
        END SELECT
    END Cliente

TASK BODY Empleado is
    pago:real;
    BEGIN
        loop
            accept realizarPago (pago: in real; compro
                comprobante:= cobrar(pago);
            end realizarPago;
        end loop;
    END Empleado;

    BEGIN
        null
    END Banco;
```

d) Implemente una solución donde los clientes esperan a lo sumo 10 minutos para ser
atendidos. Si pasado ese lapso no fueron atendidos, entonces solicitan atención una vez más y se retiran si no son atendidos inmediatamente.

```
Procedure Banco is

    TASK TYPE Cliente;

    TASK Empleado is
        ENTRY realizarPago(pago: in real; comprobante: out te
    End Empleado

    clientes = array (1..C) of Cliente;

    TASK BODY Cliente is
        pago: real;
        comprobante: text;
        BEGIN
            pago:= retornarMontoAPagar();
            SELECT
                Empleado.realizarPago(pago, comprobante);
            or delay (10) //espera 10 minutos a ser atendido
                SELECT
                    Empleado.realizarPago(pago, comprobante
                else
                    //intenta una vez más y si no es atendido
                END SELECT
            END SELECT
        END Cliente

    TASK BODY Empleado is
        pago:real;
        BEGIN
            loop
                accept realizarPago (pago: in real; comproban
                    comprobante:= cobrar(pago);
                end realizarPago;
            end loop;
        END Empleado;

        BEGIN
            null
```

```
        END Banco;
```

3. Se dispone de un sistema compuesto por 1 central y 2 procesos periféricos, que se
comunican continuamente. Se requiere modelar su funcionamiento considerando las
siguientes condiciones:

- La central siempre comienza su ejecución tomando una señal del proceso 1; luego
toma aleatoriamente señales de cualquiera de los dos indefinidamente. Al recibir una
señal de proceso 2, recibe señales del mismo proceso durante 3 minutos.
- Los procesos periféricos envían señales continuamente a la central. La señal del
proceso 1 será considerada vieja (se deshecha) si en 2 minutos no fue recibida. Si la
señal del proceso 2 no puede ser recibida inmediatamente, entonces espera 1 minuto y
vuelve a mandarla (no se deshecha).

```
Process Sistema is

    TASK TYPE Periferico is
        ENTRY recibirID(id: in int);}
    End Periferico

    TASK Central is
    ENTRY   recibirSeñal1();
        ENTRY recibirSeñal2();
        ENTRY finTiempo();
    End Central

    TASK Timer is
        ENTRY iniciarTiempo()
```

```
End Timer


perifericos = array (1..2) of Periferico


TASK BODY Periferico is
    idP: int
BEGIN
    accept recibirID (id: int in) do
        idP:= id
    end recibirID
    if (idP = 1 )
    BEGIN
        loop
        SELECT
            Central.recibirSeñal1();
            or delay 120
                //desechar la señal
        END SELECT
        end loop
    END
    else
        BEGIN
            loop
            SELECT
                Central.recibirSeñal2();
                or
                    delay 60 //espera 1 minuto
                    Central.recibirSeñal2();
            END SELECT
            end loop
        END
    END Periferico

    TASK BODY Central is
        seguir: boolean;
    BEGIN
```

```
            accept recibirSeñal1() do}
                //recibe la señal del periferico 1
            end recibirSeñal1

            loop
                SELECT
                    accept recibirSeñal1() do
                        //recibe la señal del periferico1
                    end recibirSeñal1();
                    or
                        accept recibirSeñal2() do
                            //recibir la señal del periferico
                        end recibirSeñal2();
                        Timer.iniciarTiempo()
                        seguir:= true;
                        while (seguir) begin
                            SELECT
                                accept recibirSeñal2() do
                                    //recibir la señal del pe
                                end recibirSeñal2()
                                or when (finTiempo'count != 0
                                    accept finTiempo();
                                    seguir:= false
                            END SELECT
                        END
                    END SELECT
                END LOOP
            END Central

            TASK BODY Timer is
            BEGIN
                LOOP
                accept iniciarTiempo() do
                end;
                delay 180
                Central.finTiempo();
                END loop
            END Timer
```

```
BEGIN
    for i:= 1 to 2
        perifericos(i).recibirID(i)
END Sistema
```

4. En una clínica existe un médico de guardia que recibe continuamente peticiones de
atención de las E enfermeras que trabajan en su piso y de las P personas que llegan a la
clínica ser atendidos.
Cuando una persona necesita que la atiendan espera a lo sumo 5 minutos a que el médico lo
haga, si pasado ese tiempo no lo hace, espera 10 minutos y vuelve a requerir la atención del
médico. Si no es atendida tres veces, se enoja y se retira de la clínica.
Cuando una enfermera requiere la atención del médico, si este no lo atiende inmediatamente
le hace una nota y se la deja en el consultorio para que esta resuelva su pedido en el
momento que pueda (el pedido puede ser que el médico le firme algún papel). Cuando la
petición ha sido recibida por el médico o la nota ha sido dejada en el escritorio, continúa
trabajando y haciendo más peticiones.

   El médico atiende los pedidos dándole prioridad a los enfermos que llegan para ser atendidos.
Cuando atiende un pedido, recibe la solicitud y la procesa durante un cierto tiempo. Cuando
está libre aprovecha a procesar las notas dejadas por las enfermeras

```
Procedure Clinica

TASK Medico is
    ENTRY atenderPersona;
    ENTRY atenderEnfermera;
End Medico

TASK Administrador is
    ENTRY dejarNota(nota: IN Text);
    ENTRY verNotas(notas: OUT cola);
END Administrador;

TASK TYPE Enfermera
TASK TYPE Persona



personas= array (1..P) of Persona;
enfermeras= array (1..N) of Enfermera;



TASK BODY Medico is
    notas: cola;
    BEGIN
        loop
        BEGIN
                SELECT
                 ACCEPT atenderPersona() do
                     //atiende a la persona
                    END atenderPersona
                or
                    WHEN(atenderPersona'count == 0) =>
                        ACCEPT atenderEnfermera() do
                            //atiende a la enfermera
                        END atenderEnfermera
                    END SELECT
                    Administrador.verNotas(notas)
```

```
        END lopp;
    END Medico;



TASK BODY Persona is
BEGIN
    SELECT
        Medico.atenderPersona();
        or delay(5)
                for i:= 1 to 3
                    SELECT
                    Medico.atenderPersona();
                    break;
                    or delay(5)
                        null;
                END SELECT;
    END SELECT;
END Persona;

TASK BODY Enfermera is
    nota: text;
BEGIN
    loop
    Begin
        SELECT
            Medico.atenderEnfermera();
            else
                Administrador.dejarNota(nota);
    END loop;
END Enfermera

TASK BODY Administrador is
    nota: text;
    notas: cola;
    BEGIN
        loop
                SELECT
```

```
                ACCEPT dejarNota(nota: in text) do
                    notas.push(nota);
                end dejarNota
            or
                    ACCEPT verNotas(notasDejadas: out cola) d
                        for i:= 1 to notas.length()
                            notasDejadas.push(notas.pop);
                    END verNotas;
            END SELECT
        END loop;
    END Administrador

BEGIN
    null;
End Clinica
```

5. En un sistema para acreditar carreras universitarias, hay UN Servidor que atiende pedidos
de U Usuarios de a uno a la vez y de acuerdo con el orden en que se hacen los pedidos.
Cada usuario trabaja en el documento a presentar, y luego lo envía al servidor; espera la
respuesta de este que le indica si está todo bien o hay algún error. Mientras haya algún error,
vuelve a trabajar con el documento y a enviarlo al servidor. Cuando el servidor le responde
que está todo bien, el usuario se retira. Cuando un usuario envía un pedido espera a lo sumo
2 minutos a que sea recibido por el servidor, pasado ese tiempo espera un minuto y vuelve a
intentarlo (usando el mismo documento).

```
Procedure Sistema is

    TASK Servidor is
        ENTRY
```

```
END Servidor

TASK TYPE Usuario

usuarios= array (1.. U) of Usuario;

TASK BODY Servidor is
BEGIN
    loop
            accept chequearDocumento(doc: IN text; error:
                error:= chequearErrores(doc)
            end chequearDocumento;
    end loop
END Servidor

TASK BODY Usuario is
    error: boolean;
    doc: text;
    BEGIN
        error:= true;
        while(error) loop
            trabajarDocumento(doc);
            SELECT
                Sistema.chequearDocumento(doc, error);
                or delay(2)
                    delay(1)
                    Sistema.chequearDocumento(doc, error)
            END Select;
        end loop;
    END Usuario;
```

6. En una playa hay 5 equipos de 4 personas cada uno (en total son 20 personas donde cada
una conoce previamente a que equipo pertenece). Cuando las personas van llegando
esperan con los de su equipo hasta que el mismo esté completo (hayan llegado los 4

integrantes), a partir de ese momento el equipo comienza a jugar. El juego consiste en que
cada integrante del grupo junta 15 monedas de a una en una playa (las monedas pueden ser
de 1, 2 o 5 pesos) y se suman los montos de las 60 monedas conseguidas en el grupo. Al
finalizar cada persona debe conocer el grupo que más dinero junto. Nota: maximizar la
concurrencia. Suponga que para simular la búsqueda de una moneda por parte de una
persona existe una función Moneda() que retorna el valor de la moneda encontrada.

```
Procedure Playa is

    TASK TYPE Equipo is
        ENTRY barrera();
        ENTRY identificar(id: IN int);
        ENTRY sumarMonedas(totalMonedas: IN real);
        ENTRY equipoGanador(idE: OUT int);
    End Equipo

    TASK Administrador is
        recibirMontoEquipo(monto: IN real; idE: IN int);
        equipoMayorMonto(idE: OUT int);
    End Administrador;

    TASK TYPE Persona is
     ENTRY comenzar(idP: in int);


    personas= array(1..20) of Persona;
    equipos= array(1..5) of Equipo;

        TASK BODY Persona is
        idEquipo, idEganador, id: int;
        total: real;
```

```
BEGIN
    total:=0;
    accept identificarP(id: in int) do
        id:= id
        end identificar
    equipos[idEquipo].barrera(id);
    accept equipos[idEquipo].comenzar() do
        END comenzar;
    for i:= 1 to 15 do
    BEGIN
        //busca una moneda
        total+= Moneda();
    END;
    equipos[idE].sumarMonedas(total);
    equipos[idE].equipo(idEganador);
END Persona;

TASK BODY Equipo is
    id, equipoMax: int;
    totalEquipo: int;
    colaPersonas: cola;
BEGIN
    totalEquipo:=0;
    accept identificar(id: IN int) do
        id:= id
    end idenitificador

    for i:= 1 to 4 do
        accept barrera(idP: int in) do
            colaPersonas.pop(idP);
        end barrera;


    for i:= 1 to 4 do
        personas[colaPersonas.pop()].comenzar();

    for i:= 1 to 4 do
    BEGIN
```

```
            accept sumarMonedas(total: IN int) do
                totalEquipo+= total;
            end sumarMonedas;
        END
        Administrador.recibirMontoEquipo(total, id);
        Administrador.equipoMayorMonto(equipoMax);
        for i:= 1 to 4 do
            accept equipoGanador(idE: OUT int) do
                idE:= equipoMax;
            end equipoGanador;

    END Equipo;

    TASK BODY Administrador is
        equipoMax: int;
        equipos: cola;
        BEGIN
            for i:= 1 to 5 do
                accept recibirMontoEquipo(total: IN real;
                    equipos.push(id, total);
                end recibirMontoEquipo;

        calcularMontoMaximo(equipos, equipoMax) //procedi
        for i:= 1 to 5 do
            accept equipoMayorMonto(idE: OUT int) do
                idE:= equipoMax;
            end equipoMayorMonto;
    END Administrador;

    BEGIN
        for i:= 1 to 5 do
            equipos[i].identificar(i);
        for i:= 1 to 20 do
            personas[i].identificarP(i);
    END Playa
```

7. Se debe calcular el valor promedio de un vector de 1 millón de números enteros que se
encuentra distribuido entre 10 procesos Worker (es decir, cada Worker tiene un vector de
100 mil números). Para ello, existe un Coordinador que determina el momento en que se
debe realizar el cálculo de este promedio y que, además, se queda con el resultado. Nota:
maximizar la concurrencia; este cálculo se hace una sola vez.

```
Procedure Cuenta is

    TASK TYPE Worker

    TASK Coordinador IS
        darPromedio(promedio: in real)
    End Coordinador

    workers= array (1..10) of Worker

    TASK BODY Worker is
        vectorNros: vector;
        total: real;
    BEGIN
        total:= 0
        for i:= 1 to 100000
            total+= vectorNros[i]
        Coordinador.darPromedio(total/100000)
    END Worker

    TASK BODY Coordinador is
        prom: real;
    BEGIN
        for i:= 1 to 10
        begin
            accept darPromedio (promedio: in real) do
                prom+=promedio
```

```
            end darPromedio
        end
        prom:= prom/10
    END Coordinador
BEGIN
    null
END Cuenta
```

8. Hay un sistema de reconocimiento de huellas dactilares de la policía que tiene 8 Servidores
para realizar el reconocimiento, cada uno de ellos trabajando con una Base de Datos propia;
a su vez hay un Especialista que utiliza indefinidamente. El sistema funciona de la siguiente
manera: el Especialista toma una imagen de una huella (TEST) y se la envía a los servidores
para que cada uno de ellos le devuelva el código y el valor de similitud de la huella que más
se asemeja a TEST en su BD; al final del procesamiento, el especialista debe conocer el
código de la huella con mayor valor de similitud entre las devueltas por los 8 servidores.
Cuando ha terminado de procesar una huella comienza nuevamente todo el ciclo. Nota:
suponga que existe una función Buscar(test, código, valor) que utiliza cada Servidor donde
recibe como parámetro de entrada la huella test, y devuelve como parámetros de salida el
código y el valor de similitud de la huella más parecida a test en la BD correspondiente.
Maximizar la concurrencia y no generar demora innecesaria.

```
Procedure Sistema is
```

```
TASK Especialista is
    ENTRY recibirHuella(test: IN elem)
    ENTRY resultado(valorSimilitud: IN int; codigo IN int
END Especialista

TASK TYPE Servidor;

servidores= array(1..8) of Servidor;

TASK BODY Especialista is
    test: elem;
    código, valorSimilitud: int;
    codigoMax, valorMax: int;
    BEGIN
     loop
         valorMax:= -9999;
         for i:= 1 to 16 do
                     SELECT
                         accept resultado(valorSimilitud:
                             if (valorSimilitud > valorMa
                             BEGIN
                                 valorMax:= valorSimilitu
                                 codigoMax:= codigo;
                                END
                            END resultado
                        END SELECT
                    or
                        test:= tomarImagen();
                        accept recibirHuella(test: out el
                            test:= test
                        end recibirHuella
                END SELECT
        END
END Especialista;

TASK BODY Servidor is
    codigo, valorSimilitud: int;
    test: elem
```

```
        BEGIN
            loop
                    Especialista.recibirHuella(test);
                    Buscar(test, codigo, valorSimilitud);
                    Especialista.resultado(codigo, valorSimil
            end loop;
        END Servidor;

    BEGIN
        null
    END Sistema;
```

9. Una empresa de limpieza se encarga de recolectar residuos en una ciudad por medio de 3
camiones. Hay P personas que hacen reclamos continuamente hasta que uno de los camiones pase por su casa. Cada persona hace un reclamo y espera a lo sumo 15 minutos a que llegue un camión; si no pasa, vuelve a hacer el reclamo y a esperar a lo sumo 15 minutos a que llegue un camión; y así sucesivamente hasta que el camión llegue y recolecte los residuos. Sólo cuando un camión llega, es cuando deja de hacer reclamos y se retira. Cuando un camión está libre la empresa lo envía a la casa de la persona que más reclamos ha hecho sin ser atendido. Nota: maximizar la concurrencia.

```

```