Programación concurrente

Práctica 5 - Rendezvous (ADA)

1)

a)

```
1) a)

PROCEDURE Ejercicio_1-a
TASK Puente IS
     ENTRY solcitarAcceso(peso: IN integer);
     ENTRY liberarAcceso(pesoALiberar: IN integer);
END Puente;
TASK Auto;
TASK Camioneta;
TASK Camion;

TASK BODY Puente IS
     pesoActual:= 0: integer;
BEGIN
     LOOP
          SELECT
               ACCEPT liberarAcceso(pesoALiberar: IN integer) do
                    pesoActual := pesoActual - pesoALiberar;
               END liberarAcceso;
          OR
               WHEN (pesoActual + 1 <= 5) -> ACCEPT
solcitarAccesoAuto() do
                    pesoActual := pesoActual + 1;
               END solcitarAcceso;
          OR
               WHEN (pesoActual + 2 <= 5) -> ACCEPT
solcitarAccesoCamioneta() do
                    pesoActual := pesoActual + 2;
```

```
                END solcitarAcceso;
          OR
                WHEN (pesoActual + 3 <= 5) -> ACCEPT
solcitarAccesoCamion() do
                     pesoActual := pesoActual + 3;
                END solcitarAcceso;
          END SELECT;
     END LOOP;
END Puente;

TASK BODY Auto IS
     peso: integer;
BEGIN
     Puente.solcitarAccesoAuto();
     pasar();
     Puente.liberarAcceso(1);
END Auto;

arrayVehiculos: array [1..A] of Auto;

TASK BODY Camioneta IS
     peso: integer;
BEGIN
     Puente.solcitarAccesoCamioneta();
     pasar();
     Puente.liberarAcceso(2);
END Camioneta;

arrayVehiculos: array [1..B] of Camioneta;

TASK BODY Camion IS
```

```
      peso: integer;
BEGIN
      Puente.solcitarAccesoCamion();
      pasar();
      Puente.liberarAcceso(3);
END Camion;

arrayVehiculos: array [1..C] of Camion;

BEGIN
      null;
END Ejercicio_1-a;
```

b)

```
1) b)


...
OR
      WHEN (pesoActual + 1 <= 5 and solcitarAccesoCamion'count ==
0) -> ACCEPT solcitarAccesoAuto() do
            pesoActual := pesoActual + 1;
      END solcitarAcceso;
OR
      WHEN (pesoActual + 2 <= 5 and solcitarAccesoCamion'count ==
0) -> ACCEPT solcitarAccesoCamioneta() do
            pesoActual := pesoActual + 2;
      END solcitarAcceso;
...
```

2)

a)

```
2) a)

PROCEDURE ejercicio_2 IS
TASK Empleado IS
     ENTRY pagar(pago: IN Pago, comprobante: IN Comprobante);
END Empleado;

TASK TYPE Cliente;

TASK BODY Empleado IS
BEGIN
     FOR [_ := 1..C] DO
          ACCEPT pagar(pago: IN Pago, comprobante: IN
Comprobante) DO
               comprobante := procesarPago(pago);
          END pagar;
     END FOR;
END Empleado;

TASK BODY Cliente IS
     comprobante: Comprobante;
     pago: Pago;
BEGIN
     pago := getPago();
     Empleado.pagar(pago, comprobante);
END Cliente;
```

```
clientes: array [1..C] of Cliente;

BEGIN
     null;
END ejercicio_2;
```

b)

```
2) b)

...
     SELECT
          Empleado.pagar(pago, comprobante);
     OR DELAY 10*60
          null;
     END SELECT;
...
```

c)

```
2) c)

...
     SELECT
          Empleado.pagar(pago, comprobante);
     ELSE
          null;
     END SELECT;
...
```

d)

```
2) d)

...
     SELECT
          Empleado.pagar(pago, comprobante);
     OR DELAY 10*60
          SELECT
               Empleado.pagar(pago, comprobante);
          ELSE
               null;
          END SELECT;
     END SELECT;
...
```

3)

```
3)

PROCEDURE ejercicio_3 IS
TASK Central IS
     ENTRY señal_uno(s: IN Señal);
     ENTRY señal_dos(s: IN Señal);
END Central;

TASK TYPE Periferico IS
     ENTRY identificar(idPeriferico: IN integer);
END Periferico;

TASK BODY Central IS
     señal: Señal;
```

```
     tiempoSeñalDos: Time;
BEGIN
     ACCEPT señal_uno(s: IN Señal) DO
          señal:= s;
     END señal_uno;
     procesar_señal(señal);
     LOOP
          SELECT
               WHEN (now() - tiempoSeñalDos >= 3*60) => ACCEPT
señal_uno(s: IN Señal) DO
                    señal:= s;
               END señal_uno;
               procesar_señal(señal);
          OR
               ACCEPT señal_dos(s: IN Señal) DO
                    tiempoSeñalDos:= now();
                    señal:= s;
               END señal_dos;
               procesar_señal(señal);
          END SELECT;
     END LOOP;
END Central;

TASK BODY Periferico IS
     id: integer;
     señal: Señal;
BEGIN
     ACCEPT identificar(idPeriferico: IN integer) DO
          id:= idPeriferico;
     END identificar;
```

```
     LOOP
          señal:= getSeñal();
          IF (id == 1) DO
               SELECT
                    Central.señal_uno(señal);
               OR DELAY 2*60
                    null;
               END SELECT;
          ELSE
               SELECT
                    Central.señal_dos(señal);
               ELSE
                    delay(60)
                    Central.señal_dos(señal);
               END SELECT;
          END IF;
     END LOOP;
END Central;

perifericos: array [1..2] of Periferico;

BEGIN
     perifericos[1].identificar(1);
     perifericos[2].identificar(2);
END ejercicio_3;
```

4)

```
4)

PROCEDURE ejercicio_4 IS
TASK Consultorio IS
     ENTRY dejarNota(p: IN Pedido);
END Consultorio;

TASK Medico IS
     ENTRY atendeme(Pedido: IN pedido);
     ENTRY dameBola(pedido: IN Pedido);
     ENTRY notita(pedido: IN Pedido);
END Medico;

TASK TYPE Enfermera;

TASK TYPE Persona;

TASK BODY Consultorio IS
     notas: Cola<Pedido>;
BEGIN
     LOOP
          SELECT
               ACCEPT dejarNota(p: IN Pedido) DO
                    notas.push(p);
               END dejarNota;
          ELSE
               SELECT
                    Medico.notita(notas.pop);
               ELSE
```

```
                    null;
               END SELECT;
          END SELECT;
     END LOOP;
END Medico;

TASK BODY Medico IS
BEGIN
     LOOP
          SELECT
               ACCEPT atendeme(Pedido: IN pedido) DO
                    atender(pedido);
               END atendeme;
          OR
               WHEN (atendeme'count == 0) => ACCEPT
dameBola(pedido: IN Pedido) DO
                    resolver(pedido);
               END dameBola;
          OR
               WHEN (atendeme'count == 0 AND dameBola'count == 0)
=> ACCEPT notita(pedido: IN Pedido) DO
                    resolver(pedido);
               END notita;
          END SELECT;
     END LOOP;
END Medico;

TASK BODY Enfermera IS
     pedido: Pedido;
BEGIN
     LOOP
```

```
            pedido:= getPedido();
            SELECT
                  Medico.dameBola(pedido);
            ELSE
                  Consultorio.dejarNota(pedido);
            END SELECT;
      END LOOP;
END Enfermera;

TASK BODY Persona IS
      pedido: Pedido;
      atendido:= False: bool;
      intentos:= 0: integer;
BEGIN
      pedido:= getPedido();
      WHILE (integer<3 AND !atendido) DO
            SELECT
                  Medico.atendeme(pedido);
                  atendido:= True;
            OR DELAY 5*60
                  intentos ++;
                  IF (intentos < 3) DO
                        delay(10*60)
                  END IF;
            END SELECT;
      END FOR;

      IF (!atendido) DO
            enojarse("#$@%#&@#$&%");
      END IF;
END Persona;
```

```
enfermeras: array [1..E] of Enfermera;

pacientes: array [1..P] of Persona;

BEGIN
     null;
END ejercicio_4;
```

Sin número

```
Sin número

PROCEDURE ejercicio_sin_numero IS
TASK Servidor IS
     ENTRY enviarDocumento(documento: IN Documento, hayError: OUT
bool);
END Servidor;

TASK TYPE Usuario;

TASK BODY Servidor IS
BEGIN
     LOOP
          ACCEPT enviarDocumento(documento: IN Documento,
hayError: OUT bool) DO
               hayError:= procesarDocumento(documento);
          END enviarDocumento;
     END LOOP;
END Servidor;
```

```
TASK BODY Usuario IS
     documento: Documento;
     hayError:= True: bool;
BEGIN
     documento:= trabajarDocumento();
     WHILE (hayError) DO
          SELECT
               Servidor.enviarDocumento(documento, hayError);
               IF (hayError)
                    modificarDocumento(documento);
               END IF;
          OR DELAY 2*60
               delay(60);
          END SELECT;
     END WHILE;

END Usuario;

usuarios: array [1..U] of Usuario;

BEGIN
     null;
END ejercicio_sin_numero;
```

5)

```
5)

PROCEDURE ejercicio_5 IS
TASK Administrador IS
     ENTRY total(total: IN integer, nroEquipo: IN integer);
END Administrador;

TASK TYPE Equipo IS
     ENTRY llegar();
     ENTRY identificar(nroEquipo: IN integer);
     ENTRY sumar(parcial: IN integer);
END Equipo;

TASK TYPE Persona IS
     ENTRY comenzar();
     ENTRY obtenerMasRico(nro: IN integer);
END Persona;

TASK BODY Administrador IS
     totales: array [1..5] of integer;
     masRico: integer;
BEGIN
     ACCEPT total(total: IN integer, nroEquipo: IN integer) DO
          totales[nroEquipo]:= total;
     END identificar

     # Me da el índice del máximo
     masRico:= indexMax(totales);
```

```
      FOR [i:= 1..20] DO
            personas[i].obtenerMasRico(masRico);
      END FOR;
END Administrador;

TASK BODY Equipo IS
      id, total:= 0: integer;
BEGIN
      ACCEPT identificar(nroEquipo: IN integer) DO
            id:= nroEquipo;
      END identificar

      FOR [_:= 1..4] DO
            ACCEPT llegar();
      END FOR;

      FOR [i:= 1..4] DO
            Persona[i].comenzar();
      END FOR;

      FOR [_:= 1..4] DO
            ACCEPT sumar(parcial: IN integer) DO
                  total:= total + parcial;
            END sumar;
      END FOR;
      Administrador.total(total, id);
END Equipo;

TASK BODY Persona IS
      nroEquipo, parcial:=0, equipoMasRico: integer;
BEGIN
```

```
      nroEquipo:= getEquipo();
      equipos[nroEquipo].llegar();

      ACCEPT comenzar();

      FOR [_:= 1..15] DO
            parcial:= parcial + Moneda();
      END FOR;
      equipos[nroEquipo].sumar(parcial);

      ACCEPT obtenerMasRico(nro: IN integer) DO
            equipoMasRico:= nro;
      END obtenerMasRico;
END Persona;



equipos: array [1..5] of Equipo;
personas: array [1..20] of Persona;

BEGIN
      FOR [i:= 1..5] DO
            equipos[i].identificar(i);
      END FOR;
END ejercicio_5;
```

6)

```
6)

PROCEDURE ejercicio_6 IS
TASK Coordinador IS
      ENTRY sumar(suma: IN integer);
END Coordinador;

TASK TYPE Worker IS
      ENTRY enviarVector(vector: IN array of integer);
      ENTRY comenzar();
END Worker;

TASK BODY Coordinador IS
      sumaFinal:= 0: integer;
      promedioFinal: real;
BEGIN
      FOR [i:= 1..10] DO
            workers[i].comenzar();
      END FOR;

      FOR [i:= 1..10] DO
            ACCEPT sumar(suma: IN integer) DO
                  sumaFinal:= sumaFinal + suma;
            END sumar;
      END FOR;

      promedioFinal:= sumaFinal / 1_000_000;
END Coordinador;

TASK BODY Worker IS
```

```
      parteVector: array [1..100_000] of integer;
      suma:= 0: integer;
BEGIN
      ACCEPT enviarVector(vector: IN array of integer) DO
            parteVector:= vector;
      END enviarVector;

      ACCEPT comenzar();

      FOR [i:= 1..100_000] DO
            suma:= suma + parteVector[i];
      END FOR;
      Coordinador.sumar(suma);
END Worker;

workers:= array [1..10] of Worker;
vectorTotal:= array [1..1_000_000] of integer;

BEGIN
      FOR [i:= 1..10] DO
            Worker[i + 1]:= partirVector(vectorTotal, i)
      END FOR;
END ejercicio_6;
```

7)

```
7)

PROCEDURE ejercicio_7 IS
TASK Especialista IS
     ENTRY resultado(codigo: IN integer, similitud: IN integer);
END Especialista;

TASK TYPE Servidor IS
     ENTRY testear(t: IN Test);
END Servidor;

TASK BODY Servidor IS
     test: Test;
     codigo, valor: integer;
BEGIN
     LOOP
          ACCEPT testear(t: IN Test) DO
               test:= t;
          END testear;

          Buscar(test, codigo, valor);

          Especialista.resultado(codigo, valor);
     END LOOP;
END Servidor;

TASK BODY Especialista IS
     test: Test;
     codigoMax, max: integer;
```

```
BEGIN
     LOOP
          max:= -1;
          test:= getTest();
          FOR [i:= 1..8] DO
               servidores[i].testear(test);
          END FOR;
          FOR [i:= 1..8] DO
               ACCEPT resultado(codigo: IN integer, similitud: IN
integer) DO
                    IF (similitud > max) DO
                         codigoMax:= codigo;
                    END IF;
               END resultado;
          END FOR;
     END LOOP;
END Especialista;

servidores:= array [1..8] of Servidor;

BEGIN
     null;
END ejercicio_7;
```

8)

```
8)

PROCEDURE ejercicio_8 IS
TASK Administrador IS
     ENTRY reclamo(idPersona: IN integer);
     ENTRY camionLibre(idPersona: OUT integer);
END Administrador;

TASK Camion;

TASK TYPE Persona IS
     ENTRY identificar(idPersona: IN integer);
END Persona;

TASK BODY Administrador IS
     cantReclamos: array [1..P] of integer;
     personaActual: integer;
BEGIN
     FOR [i:= 1..P] DO
          cantReclamos[i]:= 0;
     END FOR;

     LOOP
          SELECT
               ACCEPT reclamo(idPersona: IN integer) DO
                    cantReclamos[idPersona] ++;
               END reclamo;
          OR
               # algunoDistintoCero devuelve True si alguna
```

```
posicion no es 0
                WHEN (algunoDistintoCero(cantReclamos)) => ACCEPT
camionLibre(idPersona: OUT integer) DO
                     # Me da el índice del máximo
                     personaActual:= indexMax(cantReclamos);
                     cantReclamos[personaActual]:= 0;
                END camionLibre;
           END SELECT;
     END LOOP;
END Administrador;

TASK BODY Camion IS
     idPersona: integer;
BEGIN
     LOOP
          Administrador.camionLibre(idPersona);

          irALaCasaDe(idPersona);
          Persona[idPersona].llegoCamion();
          recogerResiduosEnCasaDe(idPersona);
     END LOOP;
END Camion;

TASK BODY Persona IS
     atendido: bool;
     id: integer;
BEGIN
     ACCEPT identificar(idPersona: IN integer) DO
          id:= idPersona;
     END identificar;
```

```
     WHILE (!atendido) DO
          Administrador.reclamo(id);
          SELECT
               ACCEPT llegoCamion();
               atendido:= True;
          OR DELAY 15*60
               null;
          END SELECT;
     END WHILE;
END Persona;

camiones: array [1..3] of Camion;
personas: array [1..P] of Persona;

BEGIN
     FOR [i:= 1..P] DO
          Persona.identificar(i);
     END FOR;
END ejercicio_8;
```