Programación concurrente

Práctica de repaso memoria distribuida

Pasaje de mensajes

1)

a)

```
1) a)

chan imprimir(Documento);

Process Empleado [id = 0..99] {
      while(true) {
            Documento documento = getDocumento();
            send imprimir(documento);
      }
}

Process Impresora [id = 0..4] {
      Documento documento;
      while(true) {
            receive imprimir(documento);
            imprimirDocumento(documento);
      }
}
```

b)

```
1) b)

Process Empleado [id = 0..99] {
      while(true) {
            Documento documento = getDocumento();
            Coordinador!imprimir(documento);
      }
}

Process Coordinador {
      Cola<Documento> documentos;
      do
      -     Empleado[*]?imprimir(documento) -> {
                  documentos.push(documento);
            }
      -     (!documentos.empty()); Impresora[*]?listo(idImpresora)
-> {

Impresora[idImpresora]!recibirDocumento(documentos.pop());
            }

}

Process Impresora [id = 0..4] {
      while(true) {
            Coordinador!listo(id);
            Coordinador?recibirDocumento(documento);
            imprimirDocumento(documento);
      }
}
```

2)

```
2)

Process Persona [id = 0..P-1] {
      Coordinador!solicitarAcceso(id);
      Terminal?puedoPasar();
      usarTerminal();
      Terminal!liberar();
}

Process Coordinador {
      Cola<int> solicitudes;

      do:
      -     Persona[*]?solicitarAcceso(idPersona) -> {
                  solicitudes.push(idPersona);
            }
      -     (!solicitudes.empty()); Terminal?listo() -> {
                  Terminal!dejarPasar(solicitudes.pop());
            }
}

Process Terminal {
      int idPersona;

      for [_ = 0..P-1] {
            Coordinador!listo();
            Coordinador?dejarPasar(idPersona);
            Persona[idPersona]!puedoPasar();
            Persona[idPersona]?liberar();
```

```
    }
}
```

3)

```
3)

chan solicitud(Boleta[], Dinero, int),
solicitudPrioritaria(Boleta[], Dinero, int),
solicitudEmbarazada(Boleta[], Dinero, int), solicitudEnviada(),
vuelto[0..P-1](Dinero, text);

Process Persona [id = 0..P-1] {
    Dinero pago, vuelto;
    text recibos;

    Boleta[] boletas = getBoletas();

    bool estoyEmbarazada = getEmbarazo();

    if (estoyEmbarazada) {
        send solicitudEmbarazada(boletas, pago, id);
    } else if (boletas.length() < 5) {
        send solicitudPrioritaria(boletas, pago, id);
    } else {
        send solicitud(boletas, pago, id);
    }
    send solicitudEnviada();

    receive vuelto[id](vuelto, recibos);
```

```
}

Process Cajero {
     Dinero pago, vuelto;
     text recibos;
     Boleta[] boletas;
     int idPersona;

     while(true) {
          receive solicitudEnviada();
          if (empty(solicitudEmbarazada) and
empty(solicitudPrioritaria)) {
               receive solicitud(boletas, pago, idPersona);
          } else if (empty(solicitudEmbarazada)) {
               receive solicitudPrioritaria(boletas, pago,
idPersona);
          } else {
               receive solicitudEmbarazada(boletas, pago,
idPersona);
          }
          procesarPago(boletas, pago, vuelto, recibos);
          vuelto[idPersona](vuelto, recibos);
     }
}
```

Rendezvous

1)

```
1)

PROCEDURE ejercicio_1 IS
TASK Actualizador;

TASK TYPE BancoAPI IS
     ENTRY cotizar(valor: OUT real);
END BancoAPI;

TASK BODY Actualizador IS
     valores: array [1..20] of real;
BEGIN
     LOOP
          FOR [i:= 1..20] DO
               SELECT
                    bancos[i].cotizar(valores[i]);
               OR DELAY 5*60
                    valores[i]:= 0;
               END SELECT;
          END FOR;
          actualizarPagina(valores);
     END LOOP;

END Actualizador;



TASK BODY BancoAPI IS
```

```
BEGIN
     LOOP
          ACCEPT cotizar(valor: OUT real) DO
               valor:= getCotizacionActual();
          END cotizar;
     END LOOP;
END BancoAPI;

bancos: array [1..20] of BancoAPI;

BEGIN
     null;
END ejercicio_1;
```

2)

```
2)

PROCEDURE ejercicio_2 IS
TASK Cajero IS
     ENTRY solicitud(boletas: IN Boleta[], pago: IN Dinero,
vuelto: OUT Dinero, recibos: OUT text);
     ENTRY solicitudPrioritaria(boletas: IN Boleta[], pago: IN
Dinero, vuelto: OUT Dinero, recibos: OUT text);
     ENTRY solicitudAnciana(boletas: IN Boleta[], pago: IN Dinero,
vuelto: OUT Dinero, recibos: OUT text);
END Cajero;

TASK TYPE Persona;
```

```
TASK BODY Cajero IS
BEGIN
     LOOP
          SELECT
               ACCEPT solicitudAnciana(boletas: IN Boleta[],
pago: IN Dinero, vuelto: OUT Dinero, recibos: OUT text) DO
                    procesarPago(boletas, pago, vuelto,
recibos);
               END solicitudAnciana;
          OR
               WHEN (solicitudAnciana'count == 0) -> ACCEPT
solicitudPrioritaria(boletas: IN Boleta[], pago: IN Dinero,
vuelto: OUT Dinero, recibos: OUT text) DO
                    procesarPago(boletas, pago, vuelto,
recibos);
               END solicitudPrioritaria;
          OR
               WHEN (solicitudAnciana'count == 0 AND
solicitudPrioritaria'count == 0) -> ACCEPT solicitud(boletas: IN
Boleta[], pago: IN Dinero, vuelto: OUT Dinero, recibos: OUT text)
DO
                    procesarPago(boletas, pago, vuelto,
recibos);
               END solicitud;
          END SELECT;
     END LOOP;
END Cajero;

TASK BODY Persona IS
     boletas: Boleta[];
     pago, vuelto: Dinero;
```

```
      recibos: text;
      anciana: bool;
BEGIN
      boletas:= getBoletas();
      anciana:= getEsAnciana();

      IF (anciana) DO
            Cajero.solicitudAnciana(boletas, pago, vuelto,
recibos);
      ELSE IF (boletas.length() < 5) DO
            Cajero.solicitudPrioritaria(boletas, pago, vuelto,
recibos);
      ELSE
            Cajero.solicitud(boletas, pago, vuelto, recibos);
      END IF;
END Persona;

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

BEGIN
      null;
END ejercicio_2;
```

3)

```
3)

PROCEDURE ejercicio_3 IS
TASK Central IS
     ENTRY enviarCantidadVentas(cantidad: IN integer);
END Central;

TASK TYPE Sucursal IS
     ENTRY consultar(id: IN integer);
END Sucursal;

TASK BODY Central IS
     idArticulo, totalVentas: integer;
BEGIN
     LOOP
          idArticulo:= generarArticulo();
          FOR [i:= 1..100] DO
               sucursales[i].consultar(idArticulo);
          END FOR;

          totalVentas:= 0;
          FOR [i:= 1..100] DO
               ACCEPT enviarCantidadVentas(cantidad: IN integer)
DO
                    totalVentas+= cantidad;
               END enviarCantidadVentas;
          END FOR;
     END LOOP;
END Central;
```

```
TASK BODY Sucursal IS
      cantidadVentas, idArticulo: integer;
BEGIN
      LOOP
            ACCEPT consultar(id: IN integer) DO
                  idArticulo:= id;
            END consultar;

            cantidadVentas:= ObtenerVentas(idArticulo);

            Central.enviarCantidadVentas(cantidadVentas);
      END LOOP;
END Sucursal;

sucursales: array [1..100] of Sucursal;

BEGIN
      null;
END ejercicio_3;
```