Concurrente | Práctica 4 - PMA | HOJA: 1 | FECHA

1)

```
a) chan atencion(int);

Process Cliente [id: 1..N]
 {
     send atencion(id);
 }

process Empleado
 { int idCliente;

    while (true)
     { recieve atencion (idCliente);
       atenderCliente (idCliente);
     }
 }

b) Chan atencion(int);

 process Cliente [id:1..N]
  {
    send atencion (id);
  }

 proces Empleado [id:0..1]
  { int idCliente;

     while (true)
      { recieve atencion (idCliente);
        atender Cliente (idCliente);
      }
 }
```

NOTA

```
c) chan atencion(int);
   chan pedido(int);
   chan siguiente[2](int);


process Cliente[id:1..N]
{
    send atencion(id);
}


process Empleado[id:0..1]
{ int idC;

    while(true)
    {   send pedido(id);
        reciere siguiente[id](idC);

        if(idC <> -1) { atender Cliente(idC); }

            else { delay(15min); // recibija tareas admin. x 15 min
    } }
}


Process Cordinador
{ int idC, idE;
  while(true)
  { reciere pedido(idE);
    if( empty(atencion)) { idC = -1; }
    else { reciere atencion(idC);
    }
      send siguiente[idE](idC);
  }
}
```

Concurrente | Práctica 4 - PMA | HOJA 2 | FECHA

```
2) chan pedido[s] (text, int);
   chan comprobante[P] (text);

   chan buscarCaja (int);
   chan obtenerCaja[P] (int);
   chan liberarCaja (int);
   chan hayPedido (bool);
```

usamos para evitar Busy Waiting.

```
process Caja [id:0..4]
{ int idAux;
  text pago;
  text comp;

  While (true)
  { reciere pedido[id] (pago, idAux);
     comp = generarComprobante (pago);
     send comprobante[idAux] (comp);
  }
}

process Cliente [id: 0.. P-1]
{ int idCaja;
  text pago, comp;

  send buscarCaja (id);
  send hayPedido (true);
  reciere obtenerCaja [id] (idCaja);
  send pedido [idCaja] (pago, id);
  reciere comprobante [id] (comp);
  send liberarCaja (idCaja);
  send hayPedido (true);
}
```

```
process Admin
} int cantEspera [5] = [5] 0;
  int min;
  int idAux;
  bool pedido;

  while (true)
  {  reciere hayPedido(pedido);

     if ( ! empty (buscarCaja) && empty (liberarCaja))
     }
        reciere buscarCaja (idAux);
        min = cajaMasVacia (cantEspera);
        cantEspera [min] ++;
        send obtenerCaja [idAux] (min);
     }
     else
        { if ( ! empty (liberarCaja) )
             } reciere liberarCaja (idAux);
               cantEspera [idAux] --;
             }
        }
  }
}
```

Concurrente | Práctica 4 - PMA | HOJA 3 | FECHA

```
3) chan pedidos (int, text);
   chan entregas [C] (text);
   chan siguienteV (int);
   chan respuestaV[3] (int, text);
   chan comandos (int, text);

3

process Cliente [id:0.. C-1]
{ text pedido = ...;
  text plato;

  send pedidos (id, pedido);

  receive entregas [id] (plato);
}

process Admin
{ int idV;    int idC;    text p;

  while (true)
   { receive SiguienteV(idV);
     if (empty (pedidos))
       { send respuestaV[idV] (-1, "");
       }
     } else { receive pedidos (idC, p);
              send respuestaV [idV] (idC, p);
            }
   }
}
```

```
process Vendedor [id: 0..2]
{ int idC;        text p;

   while (true)
   {
      Send siguienteV(id);

      receive respuestaV[id] (idC, p);

      if (idC == -1) { delay (60..180);    // reponerBebidas();

        } else
          { send comandos (idC, p);

        }
   }
}


process Cocinero [id: 0..1]
{ int idC,    text p;        text plato;

   while (true)
    {   receive comandos (idC, p);

        plato = cocinar (p);

        send entregas [idC] (plato);
    }
}
```

NOTA

```
4) chan porDerivar (int);
   chan porCobrar (int, int);
   chan cabinas [N] (int);
   chan facturas [N] (text);
   chan hayPedido (bool);      → usamos para evitar Busy Waiting
   chan liberar (int);


process Cliente [id: 0..N-1]
{ int numCabina;   text factura;

   send porDerivar (id);
   send hayPedido (true);
   receive cabinas [id] (numCabina);
   // usarCabina (numCabina);
   ~~send liberar (numCabina);~~
   send porCobrar (id, numCabina);
   send hayPedido (true);
   receive facturas [id] (factura);
}.

process Empleado
{ bool cabina[10] = [10] true;   bool p;  int idC, numC,  text comp;
  while (true);
  { receive hayPedido (p);
    if (!empty (porCobrar))
      { receive porCobrar (idC, numC);
        cabina[numC] = true;
        comp= cobrar (idC);
        send facturas [idC] (comp);
      }
    else
    }
      if (!empty (porDerivar) && hayCabinaLibre (cabina))
      {  receive porDerivar (idC);
         numC = obtenerCabinaLibre (cabina);
         cabina [numC] = false;
         send cabinas [idC] (numC);
      }
    }
}
```

5)
```
chan hayPedido (bool);          → para evitar
                                  Busy Waiting
chan printDirector (txt);
chan printUsuarios (txt);
chan imprimir (txt);
chan estoyLibre (int);


process Usuario [id: 0..N-1]
{  txt doc;

   while (true)
   {  doc = generarDocumento ();
      send printUsuarios (doc);
      send hayPedido (true);
   }
}


process Director
{  txt doc;
   while (true)
   {  doc = generarDocumento ();
      send printDirector (doc);
      send hayPedido (true);
   }
}
```

Concurrente | Práctica 4 - PMA | HOJA 5 | FECHA

```
process Impresora [id: 0..2]
{ text doc;

  while (true)
  { ~~send estoyLibre (id)~~;

      receive imprimir (doc);

      // imprimirDoc (doc);
  }
}


process Admin
{ bool p;   txt doc;

  while (true)
  { receive haypedido (p);

    ~~receive estoyLibre (idI)~~;

    IF (! empty (printDirector))
      { receive PrintDirector (doc);}

    else
      { receive printUsuarios (doc); }

    send imprimir (doc);

  }
}
```

NOTA

Concurrente Práctica 4 - PMS

1)
a)
```
process Examinador [id: 0.. R-1]
{ txt sitio;
  while (true)
  { sitio = buscandoSitio();
     Analizador! reporte (sitio);
  }
}

process Analizador
{ txt sitio;     txt res;
  while (true)
  {   Examinador [*]? reporte (sitio);
     res = analizar (sitio);
  }
}
```

Poco eficiente
↓
Examinador no puede seguir buscando sitio hasta que el Analizador no reciba el msj.

Opción MÁS EFICIENTE → con un proceso Administrador.

```
process Examinador [id: 0.. R-1]
{ txt sitio;
  while (true)
  { sitio = buscandoSitio();
     Administrador! reporte (sitio);
} }

Process Admin
{ cola Buffer,    txt sitio;
  do   Examinador[*]? reporte(sitio) → push (Buffer, sitio);
  □ not Empty (Buffer); Analizador? siguiente() →
           Analizador! reporte (pop (buffer));
```

}

```
process Analizador
{ txt sitio;     txt res;
  while(true)
  { Admin! siguiente();
    Admin? reporte (sitio);
    res= analizar (sitio);
  }
}
```

c) Idem solución anterior. El proceso Admin respeta el orden de llegada encolando los reportes.

2)
```
process EmpleadoUno
 { txt ADN;
   while(true)
   { ADN= prepararMuestra();
     Admin! muestra (ADN);
   }
 }
```

```
process Admin
{ txt ADN;   cola Buffer;
  do EmpleadoUno? muestra(ADN); → push(Buffer, ADN);
   □ notEmpty(Buffer); Empleado2? siguiente() →
          Empleado2 ! muestra (pop(Buffer));
} }
```

```
process EmpleadoDos
{ txt ADN;  txt res;  txt set;
 while(true)
 { Admin! siguiente();
   Admin? muestra (ADN);
   set = armar set (ADN);
   EmpleadoTres ! muestra (set);
   EmpleadoTres? resultado (res);
   Archivar(res);
}}
```

```
process EmpleadoTres
{ txt set;   txt res;
  while (true)
  { EmpleadoDos ? muestra (set);
    res = analizar (set);
    EmpleadoDos ! resultado (res);
  }
}
```

NOTA

HOJA: 7
FECHA

Concurrente Práctica 4 - PMS

3) a)

```
process Alumno[id: 0..N-1]
{ txt examen= ...;     double n;
  txt res;

  res= resolver(examen);
  Admin! entregar(res, id);
  Profesor? nota(n);
}

process Admin
{ txt res;  int idA;  cola Buffer;

  do Alumno[*]? entregar(res, idA); → push(Buffer,(res, idA))
  □ notEmpty(Buffer); profesor? siguiente() →
              profesor! corregir(pop(Buffer));
  od
}

process Profesor
{ txt res;  double n;   int idA;

 while (true)
 { Admin! siguiente();
   Admin? corregir(res, idA);

   n= CorregirExamen(res);
   Alumno[idA]! nota(n);
 }
}
```

NOTA

b)
```
process Alumno [id: 0..N-1]
{ txt examen = ...;   double n;   txt res;

    res = resolver(examen);
    Admin ! entregar (res, id);
    Profesor [*] ? nota (n);
}

process Admin
{ txt res;   int idA, idP;   cola Buffer;

    do Alumno [*] ? entregar (res, idA); → push (Buffer, (res, idA));

      □ notEmpty (Buffer); profesor [*] ? Siguiente (idP); →
                    profesor [idP] ! corregir (pop (Buffer));

    od
}

process Profesor [id: 0.. P-1]
{ txt res;   double n;   int idA;

    while(true)
    { Admin ! siguiente(id);
      Admin ? corregir (res, idA);

      n = corregirExamen (res);
      Alumno [idA] ! nota (n);
    }
}
```
NOTA

Concurrente	Práctica 4 - PMS	HOJA: 8
		FECHA

```
c) process Alumno [id: 0..N-1]
 { txt examen= ...;     double n;    txt res;

     Admin ! llegue ();
     Admin ? empezar ();

     res= resolver (examen);
     Admin ! entregar (res, id);
     Profesor [*]? nota (n);
 }

 Process Admin
 { txt res;  int idA, idP;  cola Buffer;

   For (i=0... N-1) { Alumno[*]? llegue (); }

   For (i=0..N-1) { Alumnos [i]! empezar(); }

  do Alumno[*]? entregar (res, idA); → push (Buffer, (res, idA));
  □ notEmpty (Buffer) ; profesor [*] ? siguiente (idP);  →

          profesor [idP] ! corregir (pop (Buffer));
  od
 }

Process Profesor [id: 0..P-1]
{ txt res,  double n;  int idA;
  while (true)
 { Admin! siguiente (id);
   Admin? corregir (res, idA);
   n= corregirExamen (res);
   Alumno [idA]! nota (n);
}}
```

4) a)

```
process Persona[id:0..P-1]
{
    Empleado ! PideAcceso(id);
    Empleado ? esperoTurno();

    // usar Simulador();

    Empleado ! Liberar();
}

process Empleado
{ bool libre = true;      int idP;

    do libre=true; Persona[*] ? PideAcceso(idP); →
                    libre = False;
                    Persona[idP] ! esperoTurno();

    □  libre = False;  Persona[idP] ? liberar(); →
                    libre = true;

}
```

otra opción
↓
CONSULTAR

```
process Empleado
} int idP;

    While(true)
    }  persona[*] ? PideAcceso(idP);
       Persona[idP] ! esperoTurno();
       Persona [idP] ? liberar();

    }
}
```

NOTA

b)

```
process Persona [id: 0..P-1]
{
   Empleado! PideAcceso (id);
   Empleado? EsperoTurno();

   //usarSimulador();

   Empleado! liberar();
}
                                        CONSULTAR
                                      ↗

process Empleado
{  bool libre = true;   int idP;   cola Buffer;

   do Persona[*]? PideAcceso (idP); →
            if (libre = true)
              {  libre = false
                 Persona [idP]! Esperaturno();
            }
            else  {  push(Buffer (idP)); }

   □ libre = false; Persona[*]? liberar(); →

            if (empty (buffer))
              {   libre = true;
            }
            else {  idP = pop(buffer);
                    persona [idP] ! Espero Turno ();
            }
}
```

```
5) process Espectador [id: 0.. E-1]
{

     Admin! Llegada (id);
     Admin? Turno();

     // sacar Gaseosa ();

     Admin! liberar();
}


Process Admin
{ int idE;   cola Buffer;   int cant=0;   bool libre=true;


  do (cant < E); Espectador[*]? Llegada(idE); →

              if (libre= true)
              {   libre= false;
                  Espectador [idE]! Turno();
              }
              else { push (Buffer (idE)); }

  □ (cant < E); Espectador [*]? liberar (); →
                cant ++;
              if ( empty (Buffer) } libre= true;
              }else
                   { idE= pop( Buffer);
                     Espectador [idE]! Turno();
                   }
}
```