P4 – Pasaje de Mensajes

CONSIDERACIONES PARA RESOLVER LOS EJERCICIOS DE PASAJE DE MENSAJES ASINCRÓNICO (PMA):

• Los canales son compartidos por todos los procesos.
• Cada canal es una cola de mensajes; por lo tanto, el primer mensaje encolado es el primero en ser atendido.
• Por ser PMA, el send no bloquea al emisor.
• Se puede usar la sentencia empty para saber si hay algún mensaje en el canal, pero no se puede consultar por la cantidad de mensajes encolados.
• Se puede utilizar el if/do no determinístico donde cada opción es una condición boolena donde se puede preguntar por variables locales y/o por empty de canales.

```
if   (cond 1) -> Acciones 1;
□    (cond 2) -> Acciones 2;
 ....
□    (cond N) -> Acciones N;
end if
```

De todas las opciones cuya condición sea Verdadera elige una en forma no determinística y ejecuta las acciones correspondientes. Si ninguna es verdadera, sale del if/do sin ejecutar acción alguna.
• Se debe evitar hacer busy waiting siempre que sea posible (sólo hacerlo si no hay otra opción).
• En todos los ejercicios el tiempo debe representarse con la función delay.

1. Suponga que N clientes llegan a la cola de un banco y que serán atendidos por sus empleados. Analice el problema y defina qué procesos, recursos y comunicaciones serán necesarios/convenientes para resolver el problema. Luego, resuelva considerando las siguientes situaciones:

a. Existe un único empleado, el cual atiende por orden de llegada.

```
Chan cola (int)
Chan pasar [N] (string)

Process cliente [id: 1..N]
        String msj
        Send Cola (id);
        Receive Pasar [id] (msj)
        //Esperando que el empleado termine de atenderlo
        Receive Pasar [id] (msj)          //este y el rojo de abajo, son necesarios?

Process empleado
        Int id
        While true
                Receive Cola (id)
                Send pasar [id] ("Ok")
                //Atender cliente
                Send pasar [id] ("Fin")
```

b. Ídem a) pero considerando que hay 2 empleados para atender, ¿qué debe modificarse en la solución anterior?

```
Chan cola (int)
Chan pasar [N] (string)

Process cliente [id: 1..N]
        String msj
        Send Cola (id);
        Receive Pasar[id] (msj)  //si le importara quien lo atiende aca podría recibir idE
        //Esperando que el empleado termine de atenderlo
```

```
        Receive Pasar[id]  (msj)          //este y el rosa de abajo, son necesarios?

Process empleado [id: 1..2]
        Int idC
        While true
                Receive Cola (idC)
                Send pasar [idC] ("Ok")
                //Atender cliente
                Send pasar [idC] ("Fin")
```

c. Ídem c) pero considerando que, si no hay clientes para atender, los empleados realizan tareas administrativas durante 15 minutos. ¿Se puede resolver sin usar procesos adicionales? ¿Qué consecuencias implicaría?

No, no se puede. Hay que poner un proceso coordinador porque el empleado no puede preguntar si hay alguien esperando en la cola, ya que son dos empleados esperando atender a esa gente, por lo que, si hay uno esperando, ambos preguntan si hay alguien para ir a hacer tareas administrativas ante la negativa, solo uno podrá atender al que está, y el otro empleado se quedará esperando que llegue otro cliente.

```
Chan cola (int)
Chan siguiente[2] (string)
Chan pedido (int)
Chan pasar [N] (string)

Process Cliente [id: 1..N]
        String msj
        Send Cola (id);
        Receive Pasar [id] (msj) //si le importara quien lo atiende aca podría recibir idE
        //Esperando que el empleado termine de atenderlo
        Receive Pasar [id] (msj)          //este y el rosa de abajo, son necesarios?

Process Empleado [id: 1..2]
Int IdC
While (true)
        Send pedido (id)
        Receive siguiente [id] (idC)
        If (idC<>-1) then                 //si vino un id de un cliente valido
                Send pasar [idC] ("Ok")
                //atender cliente
                Send pasar [idC] ("fin")
        Else
                Delay (900)       //hacer tareas administrativas 900=60*15

Process coordinador
        idC int
        While (true)
                Receive pedido (idE)
                If (empty (cola)) then            //si no hay nadie que enviara msj para encolarse
                        IdC= -1
                Else
                        Receive cola (idC)        //si no, recibe el msj del cliente
                Send siguiente [idE] (idC)        //le manda al empleado el id del cliente a atender
```

2. Se desea modelar el funcionamiento de un banco en el cual existen 5 cajas para realizar pagos. Existen P clientes que desean hacer un pago. Para esto, cada una selecciona la caja donde hay menos personas esperando; una vez seleccionada, espera a ser atendido. En cada caja, los clientes son atendidos por orden de llegada por los cajeros. Luego del pago, se les entrega un comprobante. Nota: maximizando la concurrencia.

```
Chan cola (int)
Chan cola [5] (int)
Chan siguiente[2] (string)
Chan pagar [P] (string)
Chan comprobante [P] (string)
Chan cola_usada [P]
Chan termine [5]
Chan despertar_coordinador

Process Cliente [id: 1..P]
        Int idcaja
        String c
        Send cola (id)                  //avisa al coordinador que se quiere encolar
        Send despertar_coordinador ("ok")
        Receive cola_usada[P](idcaja)           //lo llama su cola
        //Esperando que en la caja lo terminen de atender
        Receive comprobante (c)
        //se retira

Process caja [id: 1..5]
Int IdC
String c
While (true)
        Receive siguiente [id] (idC)      //recibe el cliente que tiene que atender
        Send cola_usada [idC] (id)
        //atender cliente
        C=generarcomprobante()
        Send comprobante [idC] (c)
        Send termine (id)
        Send despertar_coordinador ("ok")

Process coordinador
        Int idC
        int idCaja
        string A
        While (true)
                Receive despetar coordinador (a)
                If not (empty (cola)) then      //si hay clientes esperando
                        Receive cola (idC)      //recibe el msj del cliente
                        Cola_elegida= cant_clientes_x_cola.min().pos()
                        cant_clientes_x_cola[cola_elegida]= cant_clientes_x_cola[cola_elegida]+1;
                        Send siguiente [cola_elegida] (idC)     //le manda a la caja el id del cliente a
atender
                Else                    //si paso despertar_coordinador y no fue un cliente, fue una caja
                                        que termino con un cliente
                        Receive termine (idCaja)
                        cant_clientes_x_cola[idCaja]= cant_clientes_x_cola[idCaja]-1;
```

3. Se debe modelar el funcionamiento de una casa de comida rápida, en la cual trabajan 2 cocineros y 3 vendedores, y que debe atender a C clientes. El modelado debe considerar que:
- Cada cliente realiza un pedido y luego espera a que se lo entreguen.
- Los pedidos que hacen los clientes son tomados por cualquiera de los vendedores y se lo pasan a los cocineros para que realicen el plato. Cuando no hay pedidos para atender, los vendedores aprovechan para reponer un pack de bebidas de la heladera (tardan entre 1 y 3 minutos para hacer esto).
- Repetidamente cada cocinero toma un pedido pendiente dejado por los vendedores, lo cocina y se lo entrega directamente al cliente correspondiente.
Nota: maximizar la concurrencia.

```
Chan pedido int, string
Chan comida [c] string
Chan cocinar int

Process cliente[id:1..C]
        String c
        Pedido p
        Send pedido (id, p)
        //esperar pedido
        Receive comida [id] (c)
        //se va

Process coordinador
        Int idC, idV
        While (true)
                Receive desocupado(idV)
                If (empty (pedido)) then          //si no hay pedidos esperando
                        IdC= -1
                Else
                        Receive pedido (idC, p) //si no, recibe el pedido del cliente
                Send siguiente [idV] (idC, p)     //le manda al vendedor el id del cliente a atender

Process Vendedor [id: 1..2]
        Int IdCliente
        String p
        While (true)
                Send desocupado (id)
                Receive siguiente [id] (idCliente, p)
                If (idCliente<>-1) then           //si vino un id de un cliente valido
                        Send cocinar (idcliente, p)
                Else
                        Delay (180)      //reponer un pack de bebidas

Process cocinero [id: 1..2]
        Int idcliente
        String p, c
        While true
                Receive cocinar (idcliente, p)
                C=cocinar(p)
                Send comida [idcliente] (c)
```

4. Simular la atención en un locutorio con 10 cabinas telefónicas, el cual tiene un empleado que se encarga de atender a N clientes. Al llegar, cada cliente espera hasta que el empleado le indique a qué cabina ir, la usa y luego se dirige al empleado para pagarle. El empleado atiende a los clientes en el orden en que hacen los pedidos, pero siempre dando prioridad a los que terminaron de usar la cabina. A cada cliente se le entrega un ticket factura. Nota: maximizar la concurrencia; suponga que hay una función Cobrar() llamada por el empleado que simula que el empleado le cobra al cliente.

```
Chan atender string
Chan pedircabina int
Chan asignarCabina [N] int
Chan pagar int
Chan ticket string

Process cliente [id 1..n]
        Int numcabina
        Send pedircabina (id)
        Send atender ("ok")
        Receive asignarcabina [id] (numcabina)
        //usarcabina (numcabina)
        Send pagar (id, numcabina)
        Send atender ("ok")
        Receive ticket [id] (string)

Process empleado
        String a
        String factura
        Int idC
        Int cabinaslibres= 10
        Int cabinaasignada
        Int cabinadevuelta
        Array cabinadisponible [10] ([10] true)          //esto se podría cambiar por una cola
        Cola esperando

        While (true)
                Receive atender (a)
                If not empty(pagar) then        //si alguien quiere pagar
                        Receive pagar (idC, cabinadevuelta)
                        //factura= cobrar()
                        Send ticket [idc] (factura)
                        If (not empty esperando) then  //si hay alguien en la cola de espera
                                Desencolar (esperando, idC)
                                Send asignarcabina [idC] (Cabinadevuelta)

                        Else                            //si no hay nadie, libera cabina
                                Cabinadisponible [cabinadevuelta]=true
                                Cabinaslibres= Cabinaslibres+1
                Else                            //si no, el que esta esperando quiere una cabina
                        If (cabinaslibres>0) then       //si hay cabinas libres
                                Receive pedircabina (idC)
                                Cabinaasignada = primertrue(cabinadisponible).pos()
                                Cabinadisponible [cabinaasignada]=false //marca ocupada
                                Cantlibres=cantlibres-1
                                Send asignarcabina [idC] (Cabinaasignada)
                        Else
```

```
                    Receive pedircabina (idC)
                    Encolar (esperando, idC)
```

5. Resolver la administración de las impresoras de una oficina. Hay 3 impresoras, N usuarios y 1 director. Los usuarios y el director están continuamente trabajando y cada tanto envían documentos a imprimir. Cada impresora, cuando está libre, toma un documento y lo imprime, de acuerdo con el orden de llegada, pero siempre dando prioridad a los pedidos del director. Nota: los usuarios y el director no deben esperar a que se imprima el documento.

```
chan imprimirDirector(texto)
chan imprimirUsuario(texto)
chan Pedido(int)
chan Siguiente[3](texto)
chan Sincronizar()

process Usuarios [id: 1..N]
        string documento
        while (true)
                //generar documento
                send imprimirUsuario (documento)
                send Sincronizar ()

process Director
        string documento
        while (true)
                //generar documento
                send imprimirDirector (documento)
                send Sincronizar ()

process Impresoras [id: 1..3]
        while (true) {
            send Pedido (id)
            receive Siguiente[id](documento)
            //imprime (documento)

process Coordinador
        int idImp
        texto archivo

        while (true)
                receive Pedido (idImp)
                receive Sincronizar()                   //si paso esto es por que alguno mando algo
                if (not empty (imprimirDirector) then   //si hay algo del director
                        receive imprimirDirector(documento)
                        send Siguiente[idImp](documento)
                else
                        receive imprimirUsuario (documento)
                        send Siguiente[idImp](documento)
```

CONSIDERACIONES PARA RESOLVER LOS EJERCICIOS DE PASAJE DE MENSAJES SINCRÓNICO (PMS):

- Los canales son punto a punto y no deben declararse.
- No se puede usar la sentencia empty para saber si hay algún mensaje en un canal.
- Tanto el envío como la recepción de mensajes es bloqueante.

• Sintaxis de las sentencias de envío y recepción:
Envío: nombreProcesoReceptor!port (datos a enviar)
Recepción: nombreProcesoEmisor?port (datos a recibir)
El port (o etiqueta) puede no ir. Se utiliza para diferenciar los tipos de mensajes que se podrían comunicarse entre dos procesos.
• En la sentencia de comunicación de recepción se puede usar el comodín * si el origen es un proceso dentro de un arreglo de procesos. Ejemplo: Clientes[*]?port(datos).
• Sintaxis de la Comunicación guardada:
Guarda: (condición booleana); sentencia de recepción → sentencia a realizar

Si no se especifica la condición booleana se considera verdadera (la condición booleana sólo puede hacer referencia a variables locales al proceso).
Cada guarda tiene tres posibles estados:
Elegible: la condición booleana es verdadera y la sentencia de comunicación se puede resolver inmediatamente.
No elegible: la condición booleana es falsa.
Bloqueada: la condición booleana es verdadera y la sentencia de comunicación no se puede resolver inmediatamente.

Sólo se puede usar dentro de un if o un do guardado:

El if funciona de la siguiente manera: de todas las guardas elegibles se selecciona una en forma no determinística, se realiza la sentencia de comunicación correspondiente, y luego las acciones asociadas a esa guarda. Si todas las guardas tienen el estado de no elegibles, se sale sin hacer nada. Si no hay ninguna guarda elegible, pero algunas están en estado bloqueado, se queda esperando en el if hasta que alguna se vuelva elegible.
El do funciona de la siguiente manera: sigue iterando de la misma manera que el if hasta que todas las guardas hasta que todas las guardas sean no elegibles.

1. Suponga que existe un antivirus distribuido que se compone de R procesos robots Examinadores y 1 proceso Analizador. Los procesos Examinadores están buscando continuamente posibles sitios web infectados; cada vez que encuentran uno avisan la dirección y luego continúan buscando. El proceso Analizador se encarga de hacer todas las pruebas necesarias con cada uno de los sitios encontrados por los robots para determinar si están o no infectados.
a) Analice el problema y defina qué procesos, recursos y comunicaciones serán necesarios/convenientes para resolver el problema.
b) Implemente una solución con PMS.

```
Process examinador [id 1..R]
        String sitio_infectado
        While (true)
                //sitio_infectado= buscar sitio infectado
                Admin ! reporte (sitio_infectado)

Process admin
        Cola fila
        String sitio
        Do
                examinador [*] ? reporte (sitio) =>
                        encolar (fila, sitio)

                not empty (fila); analizador?pedido()=>
                        (desencolar (fila, sitio)
                        analizador ! reporte (sitio))
        od
```

```
process analizador
        string sitio
        while (true)
                admin!pedido ();
                admin?reporte (sitio)
                //verificar(sitio)
```

2. En un laboratorio de genética veterinaria hay 3 empleados. El primero de ellos continuamente prepara las muestras de ADN; cada vez que termina, se la envía al segundo empleado y vuelve a su trabajo. El segundo empleado toma cada muestra de ADN preparada, arma el set de análisis que se deben realizar con ella y espera el resultado para archivarlo. Por último, el tercer empleado se encarga de realizar el análisis y devolverle el resultado al segundo empleado.

```
process Empleado1
        string muestra
        while (true) {
                muestra = PrepararMuestra()
                Admin!enviarmuestra(muestra)

process Admin []
        cola fila
        string muestra
        do
                Empleado1?enviarmuestra(muestra)=>          //si hay muestras para recibir
                        encolar(fila, m)                    //la encola
                not empty (fila) ; Empleado2 ? pedido()=>           //si el empleado 2 hizo pedido y hay
                                                                    muestras en la cola
                        Empleado2 ! trabajo(desencolar(fila, muestra)) //le manda una
        od

process Empleado2
        string muestra, set, rest
        while (true)
                Admin! pedido()
                Admin? trabajo(muestra)
                set = armarSet(muestra)
                Empleado3 ! analizar (set)
                Empleado3 ? esperaResultado(res)
                //guardarresultado (res)

process Empleado3
        string resultado, set
        while (true)
            Empleado2 ? analizar (set)
            resultado = analisis(set)
            Empleado2 ! esperaResultado(resultado)
```

3. En un examen final hay N alumnos y P profesores. Cada alumno resuelve su examen, lo entrega y espera a que alguno de los profesores lo corrija y le indique la nota. Los profesores corrigen los exámenes respectando el orden en que los alumnos van entregando.
a) Considerando que P=1.

```
Process alumno [id 1..n]
        Float nota
```

```
        String examen
        //resolver examen
        Admin ! corregir (id, examen)
        Profesor?correccion (nota)

Process admin
        Cola fila
        String examen
        Int idA
        Do
                alumno [*] ? corregir (idA, examen) =>
                        encolar (fila, (idA, examen))

                not empty (fila); profesor?siguiente() =>
                        (desencolar (fila, (idA, examen))
                        profesor ! corregi (idA, examen))
        od

process profesor
        string examen
        int idA
        float nota
        while (true)
                admin!siguiente ()
                admin?corregi (idA, examen)
                //nota = corregir (examen)
                Alumno [idA]! Corrección (nota)
```

b) Considerando que P>1.

```
Process alumno [id 1..n]
        Float nota
        String examen
        //resolver examen
        Admin ! corregir (id, examen)
        Profesor[*]?correccion (nota)

Process admin
        Cola fila
        String examen
        Int idA, idP
        Do
                alumno [*] ? corregir (examen) =>
                        encolar (fila, (idA, examen))

                not empty (fila); profesor[*]?siguiente(idP) =>
                        (desencolar (fila, (idA, examen))
                        profesor [idP]! corregi (idA, examen))
        od

process professor [id 1..P]
        string examen
        int idA
        float nota
```

```
        while (true)
                admin!siguiente (id)
                admin?corregi (idA, examen)
                //nota = corregir (examen)
                Alumno [idA]! Corrección (nota)
```

c) Ídem b) pero considerando que los alumnos no comienzan a realizar su examen hasta que todos hayan llegado al aula.
Nota: maximizar la concurrencia y no generar demora innecesaria.

```
Process alumno [id 1..n]
        Float nota
        String examen
        Admin! Llegue ()
        Admin? Empezar ()
        //resolver examen
        Admin ! corregir (id, examen)
        Profesor[*]?correccion (nota)

Process admin
        Cola fila
        String examen
        Int idA, idP
        Cant_llegaron=0
        Do
                Alumno [*] llegue () =>
                        (Cant_llegaron= Cant_llegaron+1
                        If (Cant_llegaron==n) then
                                For id:=1 to n do
                                        Alumno [id]!empezar ())
                alumno [*] ? corregir (examen) =>
                        encolar (fila, (idA, examen))

                not empty (fila); profesor[*]?siguiente(idP) =>
                        (desencolar (fila, (idA, examen))
                        profesor [idP]! corregi (idA, examen))
        od

process professor [id 1..P]
        string examen
        int idA
        float nota
        while (true)
                admin!siguiente (id)
                admin?corregi (idA, examen)
                //nota = corregir (examen)
                Alumno [idA]! Corrección (nota)
```

4. En una exposición aeronáutica hay un simulador de vuelo (que debe ser usado con exclusión mutua) y un empleado encargado de administrar su uso. Hay P personas que esperan a que el empleado lo deje acceder al simulador, lo usa por un rato y se retira. El empleado deja usar el simulador a las personas respetando el orden de llegada. Nota: cada persona usa sólo una vez el simulador.

```
Process persona [id 1..P]
```

```
        Admin ! encolarme (id)          //pide encolarse
        Empleado ? pasar ()             //espera que el empleado le de su turno
        //usar simulador
        Empleado ! termine ()           //le avisa al empleado que termino

Process admin
        Cola fila
        Int idP
        Do
                persona [*] ? encolarme (idP) =>        //recibe de cualquier persona solicitud de encolarse
                        encolar (fila, id)              //la encola
                not empty (fila); empleado?siguiente()=>//si hay alguien encolado y el empleado espera sig
                        desencolar (fila, idP)
                        empleado ! persona_sig (idP)    //le manda al siguiente
        od

process empleado
        int idP
        while (true)
                admin!siguiente ();             //le avisa al admi que le mande otro
                admin?persona_sig (idP)         //recibe ese otro del admi
                persona[idP]!pasar ()           //le avisa a ese otro que pase
                persona[idP]? Termine ()        //espera que la persona termine para pasar al siguiente
```

5. En un estadio de fútbol hay una máquina expendedora de gaseosas que debe ser usada por E Espectadores de acuerdo al orden de llegada. Cuando el espectador accede a la máquina en su turno usa la máquina y luego se retira para dejar al siguiente. Nota: cada Espectador una sólo una vez la máquina.

```
Process espectador [id 1..E]
        Admin ! encolar (id)
        Maquina ? Usar ()
        //usar maquina
        Maquina! Termine ()


Process admin
        Cola fila
        Int idE
        Do
                espectador [*] ? encolar (idE) =>
                        encolar (fila, idE)
                not empty (fila); maquina?desocupada()=>
                        (desencolar (fila, idE)
                        maquina ! siguiente (idE))
        od

process maquina
        int idE
        while (true)
                admin!desocupada ()
                admin?siguiente (idE)
                espectador [idE] usar ()
                espectador [idE] termine()
```

--*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-OTRA OPCION

```
Process espectador [id 1..E]
        Admin ! encolar (id)
        Admin ? Usar ()
        //usar maquina
        Admin ! Termine ()

Process admin
        Cola fila
        Boolean maquina_libre := true
        Int idE
        Do
                espectador [*] ? encolar (idE) =>
                        if (maquina_libre) then
                                maquina_libre:=false;
                                espectador [idE] usar ()
                        else
                                encolar (fila, idE)
                espectador [*] ? termine ()=>
                        if (not empty (fila)) then
                                (desencolar (fila, idE)
                                espectador [idE] usar ()
                        else
                                maquina_libre:=true;
        od
```