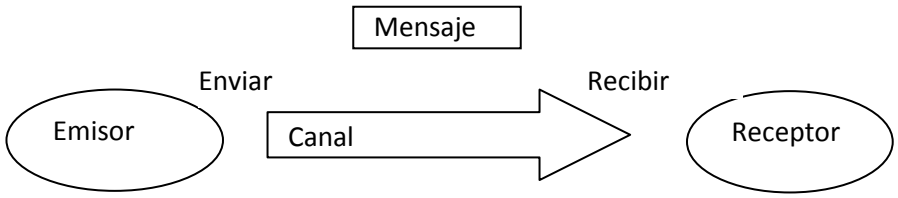
Mecanismos de paso de mensajes.

Los mecanismos de comunicación y sincronización estudiados basan su actuación en la existencia de una memoria común compartida entre procesos implicados en la comunicación. Pero cuando hablamos de sistemas o procesamiento distribuido (sistemas débilmente acoplados) hacemos referencias que no comparten memoria, reloj, etc, por lo cual no es posible una comunicación por variables compartidas. Siendo la comunicación y sincronización en estos sistemas mas compleja, la comunicación se realiza empleando redes no tan fiables incorporando problemas como la perdida de mensajes o la llegada de mensajes desordenados, la heterogeneidad posible de los nodos (puede que la comunicación tenga lugar entre diferente sistemas basados en plataformas hardware y software diferentes, diferente rendimiento, etc)

La forma natural de comunicar y sincronizar procesos en los sistemas distribuidos es mediante paso de mensajes: los procesos intercambian mensajes mediante operaciones explicitas de envío (send) y (recieve) que constituyen las primitivas básicas de comunicación de este tipo. Los sistemas de paso de mensajes podemos verlos como una extensión de semáforos en la que añadimos la comunicación de datos entre los procesos sincronizados. En el siguiente esquema de comunicación se pueden identificar los elementos que intervienen en la comunicación



Una de las ventajas de emplear mecanismos de comunicación y sincronización basados en paso de mensaje es la portabilidad de las soluciones programadas para diferentes arquitecturas de computadores, incluidos los sistemas con memoria compartida, otra ventaja es que no existe el problema del acceso en exclusión mutua a datos compartidos. Un sistema operativo o lenguaje de programación podría ofrecer herramientas, algunas basadas en memoria compartida y otras basadas en la comunicación mediante paso de mensajes, por lo que podríamos llegar a tener un mismo proceso o hilo que empleara las dos posibilidades.

Aspectos de diseño relativos a los sistemas de paso de mensajes:

1. Identificación en el proceso de comunicación
2. Sincronización
3. Características del canal (capacidad, flujo de datos, etc)

Se trata de características básicas de la comunicación y en base a ellas es posible definir esquemas de comunicación mediante paso de mensajes.

Identificación en el proceso de comunicación

Es la forma en que el emisor indica a quien va dirigido el mensaje y viceversa, es decir, la forma en que un receptor indica de quién espera un mensaje. Podemos hablar de comunicación directa o indirecta.

Un sistema de comunicación directa se caracteriza porque el emisor identifica explícitamente al receptor del mensaje en la operación de envío. El receptor, a su vez, identifica al emisor del mensaje, estableciéndose automáticamente un enlace de comunicación entre ambos. Según este esquema, las primitivas de envío/recepción tendrían la siguiente forma:

Send(A, mensaje) -> Enviar un mensaje al proceso A

Receive(B, mensaje) -> Recibir mensaje del proceso B.

La principal ventaja de este sistema de comunicación es la seguridad que ofrece en relación con la identificación de los procesos que intervienen en la transmisión, al tiempo que no introduce ningún retardo necesario para descifrar la identificación. Pero la desventaja: es que cualquier cambio que se produzca en las identificaciones de los procesos que intervienen en la comunicación obligará a modificar y recompilar el código asociado a las primitivas en las que figure el proceso o procesos renombrados.

En la comunicación indirecta no se identifica explícitamente a los procesos emisor y receptor, la comunicación se realiza colocando los mensajes en un buzon, las primitivas de comunicación es de la forma:

Send(buzonA, mensaje) -> Enviar un mensaje al buzon A

Receive(buzonA, mensaje) -> Recibir mensaje al buzon A.

Se puede usar un buzón entre 2 o mas procesos o en dos procesos podemos emplear buzones, este esquema es mas flexible, permite comunicaciones uno a uno, muchos a unos, uno a muchos y muchos a muchos. Los buzones suele llamarse puertos.

Dos cuestiones importantes en la comunicación con buzones es: 1- como se lleva a cabo la asociación de buzones a los procesos y 2- la propiedad de los mismos. En (1)los procesos declararan de antemano el buzon que van a compartir. En (2), el sistema operativo tendrá que ofrecer posibilidad para realizar tal asociación en forma dinámica, ejemplo-. Llamadas al sistema que permita conectarse y desconectarse de un buzon.

Otro esquema de comunicación indirecta es la comunicación mediante canales, en el que las operaciones de envío y recepción se realizan a través de la especificación de un canal que tiene un tipo asociado y sobre el cual se pueden enviar datos del mismo tipo. Un canal puede ser utilizado por múltiples emisores y receptores. Lenguajes como Occam y Pascal-FC soportan este esquema de comunicación.

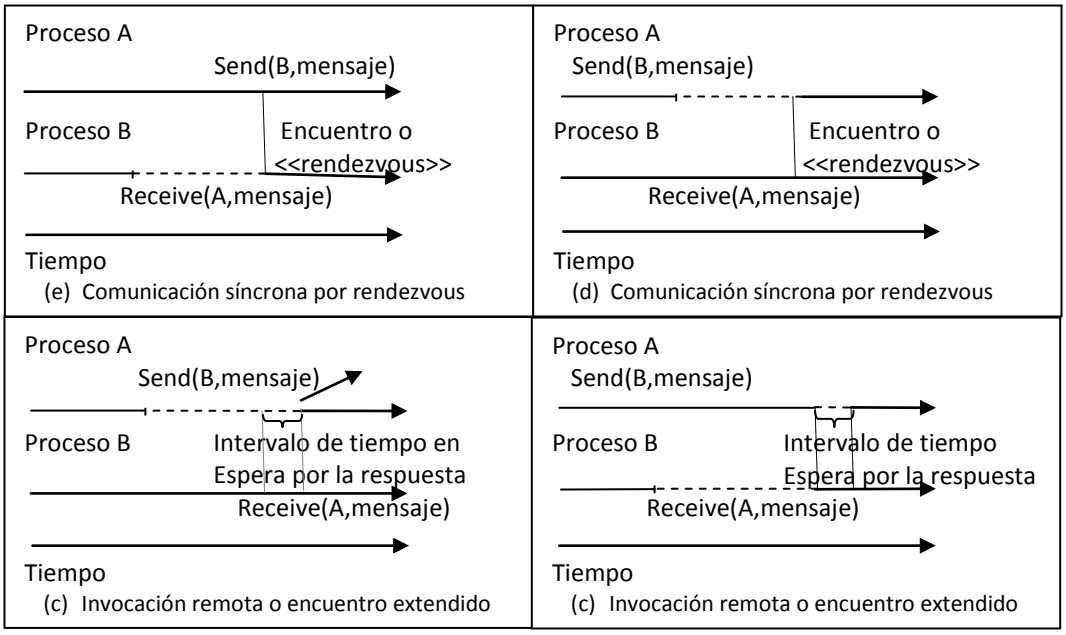
Sincronización:

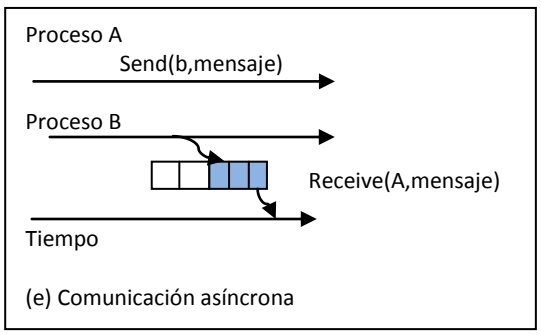
En toda comunicación, los emisores y receptores pueden no coincidir en el momento del envío y recepción del mensaje. Según el funcionamiento de las primitivas de envío y recepción, podemos hablar de comunicación:

(1) síncrona (ejemplo: la diferencia existente entre comunicación por correo electrónico o de un foro o chat), debe darse la coincidencia en el tiempo de las operaciones de envío y recepción, por lo tanto el emisor quedará bloqueado en la operación de envío send hasta que el receptor hasta que el receptor esté preparado para recibir el mensaje ejecutando la operación receive. En definitiva los procesos quedarán bloqueados hasta que se produzca el reencuentro de ambos. y

(2) asíncrona (ejemplo: la diferencia existente entre el sistema postal y el sistema telefónico), el emisor puede enviar un mensaje sin coincidir necesariamente en el tiempo de la operación de recepción,

requiriendo alojar los mensajes en el buffer hasta que el receptor los vaya retirando. En un sistema telefónico o chat la comunicación es síncrona (debe darse la coincidencia en el tiempo de las operaciones de envío y recepción), el emisor quedará bloqueado en la recepción de envío (send) hasta que el receptor este preparado para recibir el mensaje (receive) o viceversa, el receptor quedará bloqueado en la operación en la operación de recepción (receive) hasta que el emisor envíe el mensaje. "Los procesos quedarán bloqueados hasta que se produzca la coincidencia de ambos (**rendezvous**)". Cuando el receptor realiza la recepción, el emisor quedará desbloqueado y podría continuar. A veces sucede que el emisor espera la recepción del mensaje por parte del recepto y una respuesta determinada, este esquema se conoce como "extended rendezvous" o "encuentro extendido" o "invocación remota"





Como en la comunicación síncrona las primitivas Send y Receive provocan el bloqueo del emisor y receptor, también se conocen como **llamada y recepción bloqueante** (**la llamada y recepción no bloqueante tiene lugar en la comunicación asíncrona**).

Sin embargo, en la comunicación asíncrona puede llegar a ser bloqueante si se presenta el caso de buffer intermedio donde se almacenan los mensajes tengan un tamaño finito y el envío contínuo de mensajes llena dicho buffer

Es sencillo simular un esquema de comunicación síncrono a partir de un esquema asíncrono.

```
Proceso A
….
Send(B, mensaje)
Receive(B, reconocimiento)
…
```

```
Proceso B
….
Receive (A, mensaje)
send(A, reconocimiento)
…
```

Si bien se podrían dar cualquier combinación de los esquemas, las más habituales serían: llamada y recepción bloqueante, llamada y recepción no bloqueantes y la combinación bastante útil que es llamada no bloqueante y recepción bloqueante (ejemplo: un proceso emisor envía unos datos para imprimir a un proceso servidor de impresión, este proceso se podría realizar mientras el emisor continúa haciendo otras operaciones)

Las primitivas de emisión y recepción bloqueantes son mas fáciles de implementar pero son menos flexibles, el emisor tiene la confirmación que el receptor recibió el mensaje. En la recepción bloqueante el problema puede venir si el emisor que debe enviar el mensaje no lo envía por algún motivo, en este caso el receptor quedaría bloqueado indefinidamente, para evitar esto se establece un tiempo de espera en la recepción, y si pasado el tiempo y no se recibo el mensaje, el receptor se desbloquea y continúa su ejecución. En este caso la primitiva receive en un esquema de comunicación indirecta tiene la forma:

Receive (buzonA, mensaje, tiempo_de_espera)

Canal de comunicación y mensajes

Características con relación al canal de comunicación y los mensajes que fluyen por el

1- Flujo de datos: se puede establecer una comunicación :

   a- Unidireccional: la información fluye siempre en un solo sentido entre los dos interlocutores (ejemplo: correo electrónico).

   b- Bidireccional: la información podría ir en los dos sentidos (ejemplo: chat)

En las comunicaciones asíncronas la comunicación es unidireccional, el emisor emite mensaje y continúa normalmente. En la comunicación síncrona es posible las dos opciones: en ADA la comunicación es bidireccional y en OCCAM es unidireccional.

2- Capacidad del canal: hacemos referencia a la posibilidad que tiene el enlace de comunicación de almacenar los mensajes enviados por el emisor cuando el receptor no los recogió inmediatamente. Tenemos tres tipos de canales:

   a. Canal de capacidad cero: no tienen un buffers donde almacenar los mensajes (en las comunicaciones síncronas)

   b. Canal de capacidad finita: existe un buffers de tamaño limitado donde se almacenan temporalmente los mensajes enviados por el emisor, en caso de estar lleno, el emisor quedará bloqueado hasta que se libere espacio.(ejem: en comunicaciones asíncronas)

c. Canal de capacidad infinita: si la capacidad fuera a priori infinita, el sistema podría colapsar por envío continuo de mensajes por uno o varios emisores.

3- Tamaño de mensajes: la longitud del mensaje a soportar por el canal puede ser:

a. Fija: generalmente de tamaño reducido, el sistema puede reservar a priori espacio para almacenar los mensajes, si el tamaño excede el tamaño permitido, el programador deberá trocear los mensajes, con la posibilidad que lleguen desordenados.

b. Variable: tiene dos partes bien diferenciadas: 1- con información de origen, destino, tamaño de mensaje, información de control (prioridad, nro de envío, etc) y el cuerpo del mensaje (información a enviar)

4- Canales con o sin tipo: algunos esquemas de comunicación exigen definir el tipo de daros que va a fluir únicamente por el canal.

5- Paso por copia o por referencia: (ídem a programación) el envío de la información entre los procesos implicados en la comunicación puede realizarse de 2 formas:

a. Paso por copia o valor: copia exacta de los datos que el emisor quiere enviar desde el espacio de direcciones del proceso emisor al espacio de direcciones del proceso receptor. Ventaja: seguridad

b. Paso por referencia envío al receptor la dirección en el espacio de direcciones del emisor donde se encuentra el mensaje, por lo que se exige que los procesos interlocutores compartan memoria común, por lo que no es aplicables a sistemas distribuidos donde los procesos se ejecutan en máquinas diferentes. Desventajas: 1. Seguridad, el receptor debe acceder al espacio de memoria del emisor con el riesgo que implica, y 2. Se debe asegurar la exclusión mutua al acceder al mensaje, ya que varios procesos podrían acceder concurrentemente a los datos. Ventajas: 1. Comunicación eficiente, al enviar la dirección donde se encuentran los datos, el tiempo necesario es menor, 2. Ahorro de memoria, al no tener que duplicar el mensaje.

Condiciones de error en los sistemas de paso de mensajes

En un sistema distribuido en el que existen un conjunto de procesos concurrentes y cooperantes que se comunican se incrementa el número de excepciones que se generan en relación con los procesos que se ejecutaran en una misma máquina, (ejemplo de errores en la comunicación). Un bus de comunicación es más fiable que las redes de comunicación por la que pueden viajar los datos en los sistemas distribuidos. Es necesario tener en cuenta la perdida de mensajes como alteración como consecuencia por ejemplo en la transmisión. En muchos casos los protocolos de red son los que se encargan de proporcionar canales seguros, incorporando mecanismos para detectar mensajes perdidos o alterados (códigos de detección de errores) y realizar los reenvíos cuando sea necesario.

Espera selectiva

Las sentencias send y receive suspende el proceso a la espera de ser atendida, por un único interactuante. Esto supone una restricción muy fuerte, ya que un servidor debe quedar a la espera de un único evento. La sentencia select suspende el proceso a la espera de múltiples eventos.
Ahora veamos una solución para el ejemplo de productor consumidor con buffer limitado empleando un esquema de comunicación mediante paso de mensajes. En un modelo de comunicación directa síncrono, las primitivas de envío y recepción tendrán la siguiente forma:

Send(A, mensaje) -> enviar un mensaje al proceso A
Receive(B, mensaje) -> recibir mensaje del proceso B

Una solución al problema Productor – Consumidor sería:

```
Proceso Productor
Repeat
…
Producir Dato
Send(Consumidor, Dato)
…
forever
```

```
Proceso Consumidor
Repeat
…
Receive(Productor, Dato)
Consumir Dato
…
forever
```

El problema de la solución planteada es el acoplamiento entre ambos procesos. Cuando el Productor produce un elemento, este no puede continuar hasta que el proceso Consumidor lo haya consumido. Sin embargo las velocidades de ejecución de los procesos no tienen que coincidir necesariamente. Para evitar el acoplamiento, se delega el control de los elementos en proceso buffer que almacene elementos producidos por el productor y servirlos al consumidor quedando de la forma:

```
Proceso Productor
Repeat
…
  Producir Dato
  Send(Buffer, Dato)
…
forever
```

```
Proceso Consumidor
Repeat
…
  Send(Buffer, Peticion)
  Receive(Buffer, Dato)
  Consumir Dato
…
forever
```

```
Proceso Buffer
Repeat
…
  Receive(Productor, Dato)
  Receive(Consumidor, Petición
  Send(Consumidor, Dato)
…
forever
```

Sin embargo, no ha sido posible el desacople total, si el productor produjera un elemento, e intentara enviar al proceso buffer, este no podrá atender la petición a menos que el consumidor pida consumir el primer elemento producido. El problema es que el buffer solo puede esperar mensajes del emisor en un tiempo dado. En las aplicaciones clientes/servidor, los procesos servidores ejecutan algún servicio en función de las peticiones que van recibiendo de procesos clientes, pero los servidores no saben en que orden se realizarán las peticiones, entonces cuando no atienden una solicitud deberían estar dispuestos a atender cualquier petición. Esto se consigue usando de la sentencia “select” que permite la espera selectiva en varias alternativas:

```
select
   receive (buzon1, mensaje);
   sentencias;
or
   receive(buzon2, mensaje);
   sentencias
or
…..
or
  receive(buzonN, mensaje)
  sentencias;
end select;
```

Funcionamiento: al llegar a la sentencia select se evalúan todas las alternativas y se escoge una de forma aleatoria entre aquellas sobre las que se haya hecho una operación de envío por parte de algún proceso. Si ningún proceso realiza un envío sobre los buzones, el proceso quedará bloqueado en el Select hasta que se produzca alguno de estos eventos.
Una variante mas potente es la sentencia select con guardas, las guardas son condiciones que se pueden añadir a cada una de las alternativas de la sentencia select:

```
…
select
    when condicion1 =>
     receive (buzon1, mensaje);
     sentencias;
or
      when condicion2 =>
      receive(buzon2, mensaje);
      sentencias;
or
…
or
       when condicionN =>
       receive(buzonN, mensaje)
    sentencias;
end select;
….
```

No todas las sentencias necesitan llevar guardas. Al llegar la sentencia select se evalúan las guardas de todas las ramas y se consideran abiertas todas aquellas alternativas cuyo guardas se hayan evaluado cierto. A partir de este momento el compartimiento es exactamente igual al select básico considerando únicamente las alternativas abiertas. La evaluación de las condiciones se realiza una única vez al llegar a la sentencia select

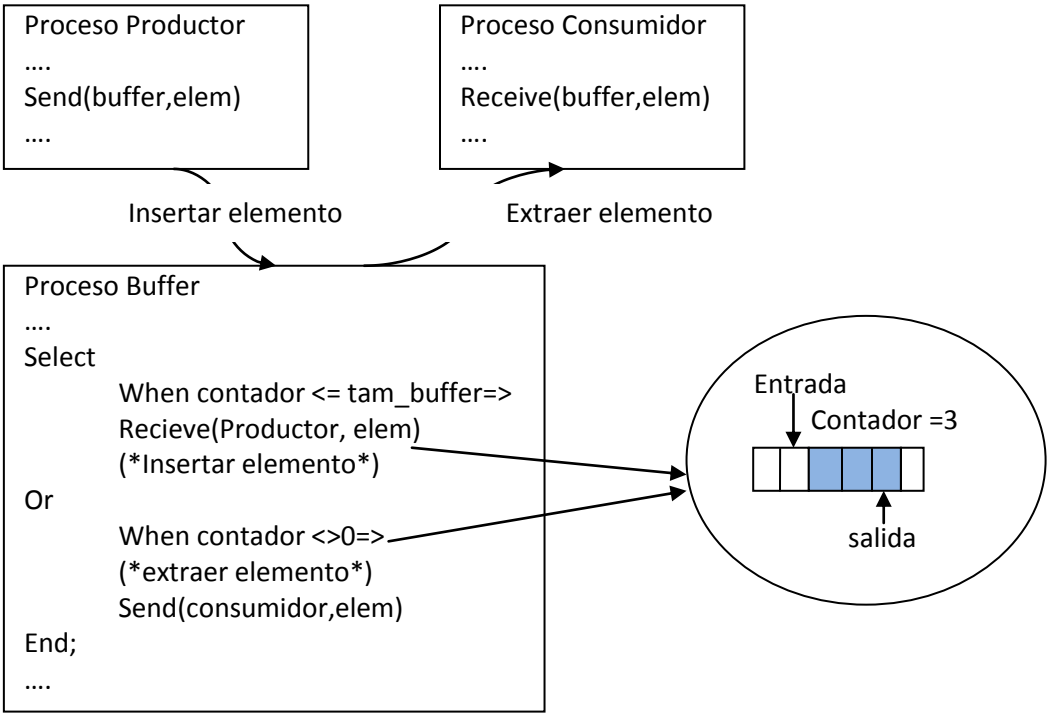
```
                Proceso P
…
select
    when condicion1 =>
     receive (buzon1, mensaje);
     sentencias;
or
      when condicion2 =>
      receive(buzon2, mensaje);
      sentencias;
or
       when condicion3 =>
       receive(buzon3, mensaje)
    sentencias;
end select;
….
```

Cuando el proceso P llega a la ejecución de la sentencia select, son ciertas las condiciones 1 y 3 pero no la condición 2, por lo que las alternativas abiertas son 1 y 3. Si ningún proceso ha enviado mensajes a los buzones 1 y 3. Entonces el proceso P quedará bloqueado. Si otros procesos envían mensajes a los buzones 1,2 y 3. Esto desbloqueará a P, ejecutándose algunas de las alternativas 1 o 3 (aleatoriamente) y esto aún si

existen mensajes en el buzón 2, y la condición 2 sea ahora cierta (decimos que la evaluación se realiza una sola vez al principio y se reevaluarán las condiciones si se vuelve a ejecutar la sentencia select)
El problema de los productores y consumidores con buffer limitado con la sentencia select:

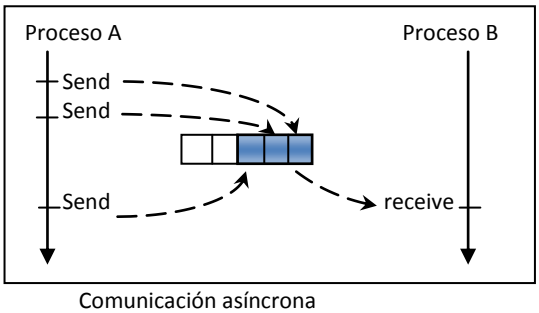
```
Program prod_cons_buffer_limitado;
Const
    Tam_buffer=4;
Process productor;
Var
    Elementoainsertar:integer;
Begin
    For mensaje := 1 to 20 do
        Send(buffer,elementoainsertar)
End;
Process consumidor;
Var
    Elementoaconsumir:integer;
Begin
    Repeat
        Receive(buffer,elementoaconsumir);
        Writeln(elementoaconsumir)
    Until mensaje=20
End;
Process buffer;
Var
    Almacen:array[1..Tam_buffer] of integer;
    Contador, entrada, salida:integer;
    Elementoainsertar:integer;
Begin
    Contador:=o;
    Entrada:=1;
    Salida:=1;
    Repeat
        Select
            When contador<=Tam_buffer =>
                Receive(productor,elementoainsertar)
                Almacen[entrada]:=elementoainsertar;
                Entrada:=(entrada mod Tam_buffer)+1;
                Contador:=contador+1;
            Or
                When contador <> o=>
                    Send(consumidor, almacen[salida]);
                    Salida:=(salida mod Tam_buffer) + 1;
                    Contador:=contador-1;
            End
    Forever
End;

Begin
    Cobegin
        Productor;
        Consumidor;
        Buffer
    Coend
End.
```

En el código se lanzan 3 procesos: un productor cuyo objetivo es insertar en el buffer 20 elementos entero, un proceso Consumidor cuyo objetivo es consumir 20 elementos enteros del buffer y finalmente tenemos un proceso buffer que es el proceso encargado de controlar el buffer, es decir, de atender peticiones del proceso productor y del proceso consumidor:



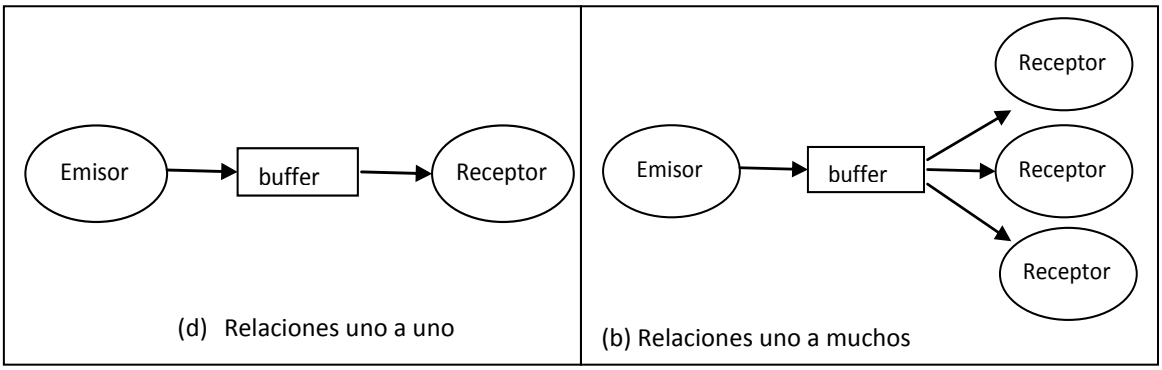
El código del proceso productor es sencillo: realiza envío al proceso controlador del buffer para que introduzca los elementos producidos (operación send). El consumidor intenta extraer elementos del buffer pidiendo al proceso controlador del buffer el siguiente elemento a consumir (operación receive). El controlador del buffer gestiona el buffer y atiende las peticiones del proceso productor y consumidor. Para ello declara el vector que va a contener elementos y 3 variables enteras para controlar el número de elementos en el buffer (contador) y por donde se realizará las inserciones (entrada) y extracciones de dicho vector (salida). Si vemos el código, cuando comienza su ejecución inicializa las variables entra a un bucle en el que espera peticiones del proceso productor o del consumidor.
El buffer no sabe el orden que van a ir realizando las peticiones de inserción o extracción. La sentencia select permite que el buffer quede esperando para recibir peticiones de insertar elementos por el productor o por peticiones del consumidor. Evitando que el productor quiera insertar un elemento cuando este lleno el buffer o que el consumidor intente consumir un elemento si el buffer esta vacío.

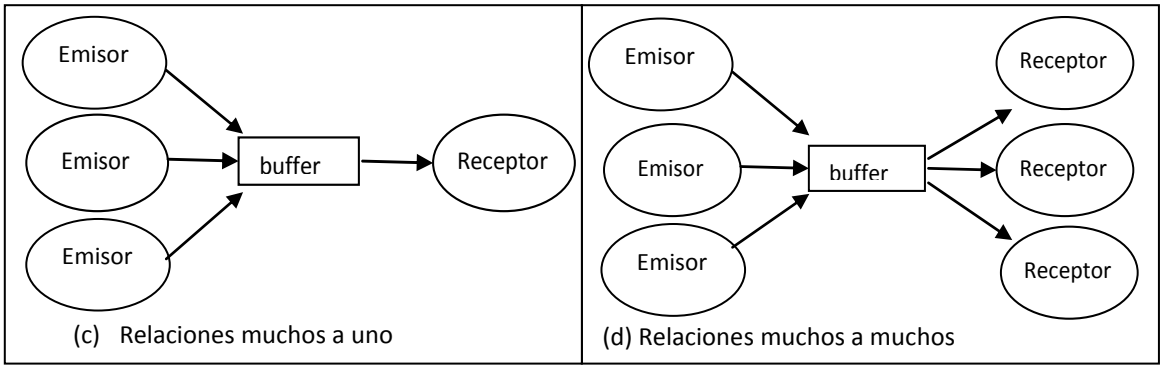
Paso de mensaje asíncrono



Comunicación asíncrona

Se pueden distinguir variantes según el comportamiento de las primitivas send y receive. Caso general: ambas operaciones no son bloqueantes, el emisor puede enviar y continuar su ejecución sin ser necesario coincidir en el tiempo de la operación de recepción y viceversa, el receptor hace una operación de recepción y continuar su ejecución aunque no exista mensajes pendientes. Mediante este esquema de comunicación es posible establecer relaciones entre emisores y receptores de tipo uno a uno, uno a muchos, muchos a uno o inclusos a muchos.





Para establecer comunicaciones entre emisores y receptores, será necesario definir un buzón de comunicaciones con la sintaxis:

Nombre_buzon: mailbox of tipo

Nombre_buzon: mailbox[1..N] of tipo

Nombre_buzon: es el identificador del buzon que será necesario emplear en el envío y recepción y tipo especifica el tipo de dato de los mensajes que pueden viajar por el. La segunda forma de declarar un buzón nos permite limitar el tamaño del buffer contenedor de mensajes, un buzon permite el envío de datos múltiples emisores a múltiples receptores (permite relaciones muchos a muchos) la operación de envío se define de la siguiente forma:

Send(b,m) enviar el mensaje m al buzon b.

El proceso emisor ejecutará la operación de envío y continuará normalmente salvo que se haya definido un buzón de capacidad limitada y este lleno, quedando el emisor bloqueado hasta que algún proceso receptor recoja un mensaje del buzón

Un proceso receptor que quiera recibir un mensaje del buzon b ejecurá la operación:

receive(b,m) Recibir del buzon b un mensaje y almacenarlo en m.

el receptor quedará bloqueado si no existen mensajes pendientes en la cola asociada al buzon (recepción bloqueante). Cabe la posibilidad de un servidor de peticiones de disco con prioridad, donde se sirvan las peticiones a través de diferentes buzones, el servidor pordria chequear el estado de diferentes buzones sin quedarse bloqueado en el primero que este vacío mediante:

EMPTY(B) devuelve verdadero si el buzón b está vacío y falso en caso contrario.

La condición comprobada sólo se garantiza en el instante justo en el que se ejecuta la operación puesto que el estado del buzón puede cambiar en cualquier momento.
Al existir la posibilidad que múltiples emisores y receptores coincidan en operaciones de envío y recepción a un buzón, estas operaciones se atómicas (indivisibles). Suponiendo que trabajamos con una red fiable, los mensajes llegarán en orden (según orden de los send) y sin alteraciones (la comunicaciones solventará cualquier problema en este sentido)
Ejemplo: si tenemos 2 procesos cooperativos P1 y P2 ejecutándose concurrentemente en una máquina que permite la comunicación mediante variables compartidas. Ambos procesos realizan consultas y modificaciones sobre esas variables compartidas

Procesos P1	Procesos P2
repeat	repeat
……..	……
Sección critica	Sección crítica
……..	……
forever	forever

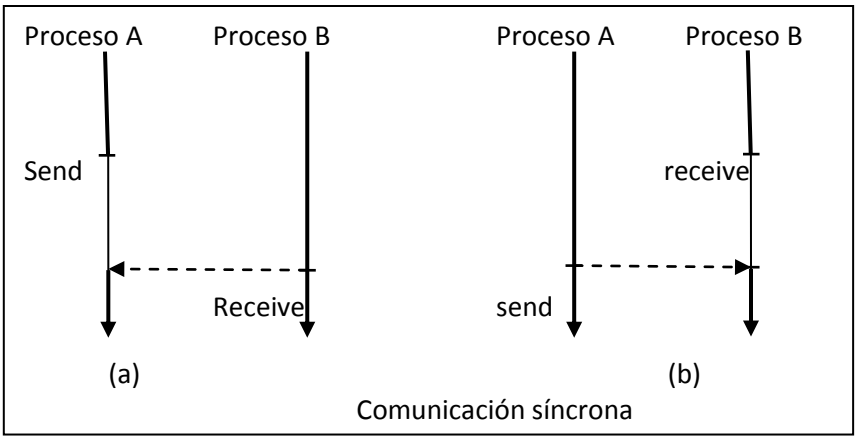
Como se puede ver en la solución propuesta (extensible a N procesos), para garantizar la exclusión mutua se emplea un buzón. Antes de lanzarse los procesos, el programa principal (padre) envía un mensaje al buzón (testigo). En este caso el contenido del mensaje en si es irrelevante porque su uso es simplemente para sincronización. Un proceso antes de entrar en su sección crítica efectúa un receive sobre el buzón. El proceso que reciba el testigo será el que ingrese a su sección crítica. Al terminar su sección, el proceso envía nuevamente el testigo al buzón, dejando libre de esta forma la entrada a cualquier otro proceso interesado en entrara en su sección crítica.

```
Program Ejemplo1
Var
    Buzon: mailbox of integer;
    Tetigo: integer;
Process P1;
Var
    testigo: integer;
begin
    repeat
        …….
        RECEIVE(buzon, testigo);
        (*Sección crítica*)
        SEND (buzon, testigo);
        …..
    Forever
End;
Process P2;
Var
    testigo: integer;
begin
    Repeat
        …….
        RECEIVE(buzon, testigo);
        (*Sección crítica*)
        SEND (buzon, testigo);
        …..
    Forever
End;
```

```
Begin
    SEND(buzon, testigo)
    Cobegin
        P1;
        P2;
    Coend
End.
```

El funcionamiento es similar al comportamiento de un semáforo binario. Para que el funcionamiento sea similar a un semáforo general basta con que el proceso padre envíe un número de mensajes iniciales (testigo) exactamente igual al valor al que se inicializaría un semáforo general.

Paso de mensaje síncrono con canales
En la comunicación síncrona, las primitivas de comunicación de envío y recepción son bloqueantes



Como se ve en la figura se puede dar dos situaciones: (a) una en la que le emisor envía un mensaje y el receptor no haya realizado aún la operación de recepción, por lo que el emisor quedará bloqueado hasta que el receptor realiza dicha operación. Y (b) el receptor realiza una operación realiza una operación de recepción y el envío aún no se realizó, quedando el receptor bloqueado hasta que el emisor realice el envío.
Cuando el emisor y el receptor continúan después de la operación de envio y recepción, tienen la certeza de que la comunicación entre ambos se ha establecido y que el mensaje ha sido intercambiado, no será necesario tener buffer de mensajes haciendo que este mecanismo sean mas simples de implementar.
El modelo de comunicación síncrona con canales permite establecer un enlace entre dos procesos, de forma que las primitivas en envío y recepción especifican el canal al que se enviará el mensaje o del que se recibirá, siendo ambas operaciones bloqueantes:

ch ! s envía el valor de la expresión s al canal ch
ch ? r recibe del canal ch un valor y lo asigna a la variable r

El resultado de ambas operaciones puede ser considerado como una asignación distribuida donde el valor de la expresión s en un proceso es asignado a otra variable de otro proceso (r=s). Los canales permiten establecer relaciones de comunicación uno a uno entre un único emisor y único receptor, estableciéndose un flujo de datos unidireccional punto a punto. Los canales suelen tener un tipo y este debe coincidir con el tipo de datos que son enviados a través del mismo.

La comunicación mediante canales exige que el canal y los datos que viajan por el sean del mismo tipo. Los canales son declarados mediante la palabra reservada cannel antes de la declaración de procesos que harán uso del mismo y el compilador es el encargado de asegurarse que los canales son usados correctamente.
Ejemplo: dos procesos que se comunican empleando este modelo de comunicación:

```
Program ejemplo
Type mensaje=
  Record
        (*estructura del mensaje*)
  End;
Var
  (* Declaración del canal, indicando el tipo de los datos que viajarán por el*)
  ch: cannel of mensaje;
Process P;
Var
  s:mensaje;
```

```
Begin
    …
    (*Envía el mensaje s al canal ch*)
    ch !s,
    …
End;

Process Q;
Var
    r: mensaje;
begin
    …
    (*Recibe un mensaje del canal ch y lo almacena en r*)
    ch ? r;
    ….
End;

begin
    cobegin
       P;
       Q
    coend
End;
```

Este esquema de comunicación es síncrono, bloqueante (en ambas operaciones) e indirecto (se realiza a través de un canal de capacidad cero), siendo necesario el encuentro del emisor y receptor en las operaciones de envío y recepción, caso contrario, el primer proceso que realice una operación sobre el canal quedará bloqueado esperando por el otro proceso.