# Pasaje de mensajes sincrónico

Los canales son de tipo link o punto a punto (es decir 1 emisor y 1 receptor)

La diferencia entre PMA y PMS es la primitiva de transmisión Send. En PMS es bloqueante:

El transmisor queda esperando que el mensaje sea recibido por el receptor. La cola de mensajes asociada a un send se reduce a un mensaje, ósea ocupa menos memoria. Naturalmente el grado de concurrencia se reduce respecto de la sincronización por PMA, ya que los emisores se bloquean.

Si bien send y send sincrónico son similares la semántica es diferente y las posibilidades de deadlock son mayores en comunicación sincrónica.

Por ejemplo, en un caso de productores y consumidores puede haber tres casos:

* Ambos tardan lo mismo en trabajar: Es igual para sincrónico y asincrónico
* El consumidor tarda más en trabajar: Produce retardo para el productor en sincrónico, pero no afecta al tiempo total porque el principal es el consumidor.
* El productor tarda más en trabajar: Produce retardo para el consumidor porque no puede seguir su ejecución hasta que el productor termina. Esto si produce una demora en el tiempo total en ambos tiempos porque el consumidor que es el que recibe mensajes está demorado por lo que tarda el productor.

Conclusión: Con PMS los pares send/receive se completarán asumiendo la demora del proceso que más tiempo consuma. Con PMA, al principio el productor es más rápido y sus mensajes se encolan y luego el consumidor descuenta tiempo consumiendo la cola de mensajes.

Para que haya una concurrencia similar a la de PMA, se debe usar un **buffer** donde guardar los mensajes y así no demorar a los procesos.

Y acá comienza, la idea de que las técnicas de PMS se basan en hacer asincrónico lo sincrónico.

La concurrencia también se reduce en interacciones cliente servidor, porque cuando un cliente está liberando un recurso no habría motivos para demorarlo hasta que el servidor reciba el mensaje, pero con PMS se puede demorar.

Otro ejemplo se da cuando un cliente quiere escribir en algo manejado por un proceso servidor. Normalmente el cliente quiere seguir inmediatamente después de un pedido de write.

Otra desventaja del PMS es la mayor probabilidad de deadlock, el programador debe ser cuidadoso de que todas las sentencias send y receive hagan matching.

# Lenguaje CSP para PMS

Fue uno de los desarrollos fundamentales en programación concurrente, muchos lenguajes reales se basan en CSP. Las ideas básicas de Hoare (el creador ) fueron PMS y **comunicación guardada**, ósea PM con waiting selectivo.

El **canal** es un link directo entre dos procesos en lugar de un mailbox global. Son half-duplex ( de una sola dirección) y nominados. Las sentencias de entrada ? y salida ! son el único medio por el cual los procesos se comunican.

Para que se produzca la comunicación deben matchear y luego se ejecutan simultáneamente.

De forma general la sintaxis sería:

**Destino!port**

**Fuente?port**

Donde Destino y Fuente nombran un proceso simple, o un elemento de un arreglo de procesos. Fuente puede nombrar a cualquier elemento de un arreglo, si se usa el comodín (\*) no se respeta ningún orden para recibir un mensaje

port son etiquetas que se usan para distinguir entre distintas clases de mensajes que un proceso podría recibir.

Dos procesos se comunican cuando ejecutan sentencias de comunicación que hacen matching (un ! y un ? al mismo canal)

Comunicación guardada

? y ! nos limitan bastante ya que son bloqueantes. Por ejemplo tendríamos problemas si un proceso quiere comunicarse con otros sin conocer el orden en que los otros quieren hacerlo con él. Puede hacerse con un comodín [\*] pero aun así todos tendrían que usar el mismo pedido, entonces estamos en la misma.

Lo que podemos hacer es **guardar** las operaciones de comunicación, es decir hacer un await hasta que una condición sea verdadera. Osea, usar el do e if de alternativa múltiple. Los do e if de CSP usan los comandos guardados de Dijkstra B 🡪 S.

Se acuerdan el usar el do para guardar mensajes sincrónicos en una cola como si fuesen asincrónicos? Bueno volvió.

Descripción formal:

**B; C 🡪 S**

B puede omitirse asumiendose true. B y C conforman la guarda y esta tiene éxito si B es true y ejecutar C no causa demora, falla si B es falsa o se bloquea si B es true pero C no puede ejecutarse inmediatamente.

Un caso con if:

Texto, Carta

Descripción generada automáticamente

Primero se evalúan las guardas:

* Si todas fallan, el if termina sin efecto
* Si al menos una tiene éxito, elige una no deterministica.
* Si algunas se bloquean, espera a que alguna de ellas tenga éxito.

Segundo, luego de elegir una guarda exitosa se ejecuta la sentencia de comunicación de la guarda elegida.

Tercero, se ejecuta la sentencia S.

**La ejecución del do es similar (se repite hasta que todas las guardas fallen)**

A partir de acá son todos ejemplos, yo los dejo por las dudas pero para mi no sirven {si sirven}

Ejemplo usando un buffer limitado

Texto

Descripción generada automáticamente

Ejemplo: Asignación de recursos

Texto

Descripción generada automáticamente

Ejemplo: Intercambio de valores

Texto, Carta

Descripción generada automáticamente

Ejemplo: Criba de eratóstenes para la generación de números primos

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Ejemplo: Ordenación de un arreglo

Imagen que contiene interior, pájaro, ave, tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente