**Clase 6 - Pasaje de Mensajes**

**Conceptos generales**

* Arquitecturas de memoria distribuida ⇒ procesadores + memo local + red de comunicaciones + *mecanismo de comunicación* / *sincronización* ⇒ **intercambio de mensajes.**
* **Programa distribuido**: programa concurrente comunicado por mensajes. Supone la ejecución sobre una arquitectura de memoria distribuida, aunque puedan ejecutarse sobre una de memoria compartida (o híbrida).
* **Primitivas de pasaje de mensajes**: interfaz con el sistema de comunicaciones ⇒ semáforos + datos + sincronización.
* Los procesos SOLO comparten canales (físicos o lógicos). Variantes para los canales:
  + Mailbox, input port, link.
  + Uni o bidireccionales.
  + Sincrónicos o asincrónicos.

Características  
  
 Los canales son lo único que comparten los procesos

* Variables locales a un proceso (“cuidador”).
* La exclusión mutua no requiere mecanismo especial.
* Los procesos interactúan comunicándose.
* Accedidos por primitivas de envío y recepción.

Mecanismos para el Procesamiento Distribuido:

* Pasaje de Mensajes Asincrónicos (PMA)
* Pasaje de Mensajes Sincrónico (PMS)
* Llamado a Procedimientos Remotos (RPC)
* Rendezvous

La estructura de la comunicación entre los procesos depende del patrón de interacción:

* Productores y consumidores
* Clientes y servidores
* Peers

| Cada mecanismo es más adecuado para determinados patrones |
| --- |

Relación entre mecanismos de sincronización  
  
**Semáforos** ⇒ mejora respecto de busy waiting.  
**Monitores** ⇒ combinan Exclusión Mutua implícita y señalización explícita.  
**PM** ⇒ extiende semáforos con datos.  
**RPC y rendezvous** ⇒ combina la interfaz procedural de monitores con PM implícito.

**Pasaje de Mensajes Asincrónicos**

Uso de canales en PMA

PMA ⇒ canales = colas de mensajes enviados y aún no recibidos.

Declaración de canales ⇒ chan ch (parámetros)

**Operación Send** → un proceso agrega un mensaje al final de la cola (“ilimitada”) de un canal ejecutando un send, que no bloquea al emisor:

send ch(expr1, ... , exprn);

**Operación Receive** → un proceso recibe un mensaje desde un canal con receive, que demora (“bloquea”) al receptor hasta que en el canal haya al menos un mensaje; luego toma el primero y lo almacena en variables locales:

receive ch(var1, ... , varn);

Las variables del receive deben tener los mismos tipos que la declaración del canal.

Receive es una primitiva bloqueante, ya que produce un espera. Semántica: el proceso NO hace nada hasta recibir un mensaje en la cola correspondiente al canal. NO es necesario hacer polling.

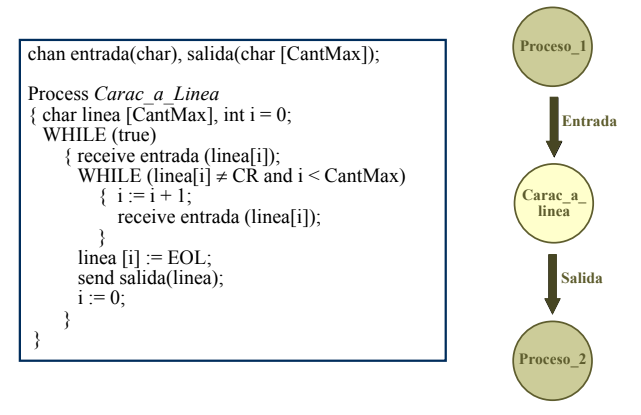
Características de los canales

* Acceso a los contenidos de cada canal: atómico y respeta orden FIFO.
* En principio los canales son ilimitados, aunque las implementaciones reales tendrán un tamaño de buffer asignado.
* Se supone que los mensajes NO se pierden ni modifican y que todo mensaje enviado en algún momento puede ser “leído”.
* **empty**(ch) → determina si la cola de un canal está vacía. Útil cuando el proceso puede hacer trabajo productivo mientras espera un mensaje, pero debe usarse con cuidado.
  + La evaluación de empty podría ser true, y sin embargo existir un mensaje al momento de que el proceso reanuda la ejecución.
  + O podría ser false, y no haber más mensajes cuando sigue ejecutando (si no en el único en recibir por ese canal).

Los canales son declarados globales a los procesos, ya que pueden ser compartidos. Según la forma en que se usan podría ser:

* Cualquier proceso puede enviar o recibir por alguno de los canales declarados. En este caso suelen denominarse mailboxes.
* En algunos casos un canal tiene un solo receptor y muchos emisores (input port).
* Si el canal tiene un único emisor y un único receptor se lo denomina link: provee un “camino” entre el emisor y sus receptores.

Ejemplo:



**Clientes y Servidores.**

Monitores Activos

* Un **Servidor** es un proceso que maneja pedidos (requerimientos) de otros procesos clientes. Veremos cómo implementar Cliente/Servidor con PMA.
* Un proceso **Cliente** que envía un mensaje a un canal de requerimientos general, luego recibe el resultado desde un canal de respuesta propio.
* En un sistema distribuido, lo natural es que el proceso **Servidor** resida en un procesador físico y M procesos **Cliente** residan en otros N procesadores (N<= M).