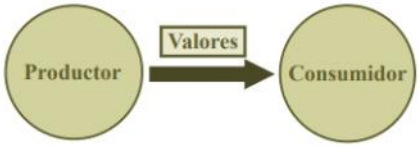
Clase 7- Pasaje de Mensajes Sincrónicos TERMINAR

jueves, 17 de octubre de 2024 08:25

Diferencias con pma:

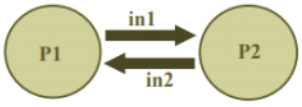
- Acá los canales son de tipo link: 1 emisor y 1 receptor
- En pms el send es bloqueante
  - El transmisor se queda esperando que el receptor lo reciba
  - La cola de mensajes asociada con un send es de 1 (menos memoria)
  - - Concurrencia que pma
- send y sync_send son parecidos pero hay más chance de deadlock en conumicación sincrónica

```
chan valores(int);

Process Productor
{  int datos[n];
   for [i=0 to n-1]
      {  #Hacer cálculos productor
         sync_send valores (datos[i]);
      }
}

Process Consumidor
{  int resultados[n];
   for [i=0 to n-1]
      {  receive valores (resultados[i]);
         #Hacer cálculos consumidor
      }
}
```



Una cagada en pms pq si uno es lento tardas una bocha, en pma se encolan los mensajes entonces desdpués el consumidor puede hacer rápido
Para lograr el mismo efecto en PMS se debe interponer un proceso "buffer".

- Dos procesos que intercambian valores.




```
chan in1(int), in2(int);

Process P1
{   int valorA = 1, valorB;
    sync_send in2(valorA);
    receive in1(valorB);
}

Process P2
{   int valorA, valorB = 2;
    sync_send in1(valorB);
    receive in2 (valorA);
}
```

```
chan in1(int), in2(int);

Process P1
{   int valorA = 1, valorB;
    sync_send in2(valorA);
    receive in1(valorB);
}

Process P2
{   int valorA, valorB = 2;
    receive in2 (valorA);
    sync_send in1(valorB);
}
```

Hay que tener cuidado pq es re facil que se bloqueen

Lenguaje para PMS

- Comunicación guardada: PM con waiting selectivo
- canal: link directo entre dos procesos en lugar de mailbox global. Son halfLas sentencias de Entrada (? o query) y Salida (! o shriek o bang) son el único medio por el cual los procesos se comunican.
  - process A { . . . B ! e; . . . } process B { . . . A ? x; . . . }
- Para que se produzca la comunicación, deben matchear, y luego se ejecutan simultáneament

Formas generales de las sentencias de comunicación:

Destino ! port(e_1, ..., e_n);

Fuente ? port(x_1, ..., x_n);

- *Destino* y *Fuente* nombran un proceso simple, o un elemento de un arreglo de procesos. *Fuente* puede nombrar *cualquier* elemento de un arreglo (*Fuente*[*]).
- ***port*** son etiquetas que se usan para distinguir entre distintas clases de mensajes que un proceso podría recibir (puede omitirse si es sólo uno).
- Dos procesos se comunican cuando ejecutan sentencias de comunicación que hacen ***matching***.

***A* ! canaluno(dato); *B* ? canaluno(resultado);**

Ejemplo de uso

Server que calcula el MCD de dos enteros con el algoritmo de Euclides. ***MCD*** espera recibir entrada en su port ***args*** desde un cliente. Envía la respuesta al port ***resultado*** del cliente.

```
Process MCD
{   int id, x, y;
    do true →
            Cliente[*] ? args(id, x, y);
            do  x > y → x := x - y;
            □   x < y → y := y - x;
            od
            Cliente[id] ! resultado(x);
    od
}
```

➢ *Cliente[i]* se comunica con *MCD* ejecutando:

```
...
MCD ! args(i, v1, v2);
MCD ? resultado(r);
...
```

- Problemas si un proceso quiere comunicarse con otros sin saber el orden en que se quieren comunicar con él
- Las operaciones de comunicación (? y !) pueden ser guardadas, es decir hacer un AWAIT hasta que una condición sea verdadera

Las sentencias de comunicación guardada soportan comunicación no determinística:

B; C →S;

- ***B*** puede omitirse y se asume *true*

Las sentencias de comunicación guardada soportan comunicación no determinística:

$$B; C \rightarrow S;$$

- ***B*** puede omitirse y se asume *true*.
- ***B*** y ***C*** forman la guarda.
- La guarda tiene éxito si ***B*** es *true* y ejecutar ***C*** no causa demora.
- La guarda falla si ***B*** es *falsa*.
- La guarda se bloquea si ***B*** es *true* pero ***C*** no puede ejecutarse inmediatamente.

```
Process Copiar2
{
   char c1, c2;
   Oeste ? (c1);  // Lee el primer carácter de Oeste
   do
      Oeste ? (c2) ? Este ! (c1); // Lee un nuevo carácter y envía el carácter previamente almacenado
      c1 = c2;                    // El nuevo carácter leído se convierte en el siguiente a enviar
   ? Este ! (c1) ? Oeste ? (c1);    // Alternativamente, si no recibe nada de Oeste, envía el carácter y
espera otro
   od
}
```

Todos los ejemplos los tengo que ver con más tiempo después :(

Paradigmas para la interacción entre procesos

Paradigma 1: master / worker Implementación distribuida del modelo Bag of Task. El que sabe trabajar es el worker, es como un cliente servidor donde el servicio que brinda el servidor es dar más tareas

Paradigma 2: algoritmos heartbeat Los procesos periódicamente deben intercambiar información con mecanismos tipo send/receive. Conozco solo a algunos procesos, no a todos. los procesos tienen que ir intercambiando información. solo voy a tener info de mis vecinos directos. en la segunda iteración mi vecino tiene data de los suyos, entonces voy conociendo ás data a más iteraciones

Paradigma 3: algoritmos pipeline La información recorre una serie de procesos utilizando alguna forma de receive/send.

Paradigma 4: probes (send) y echoes(receive) La interacción entre los procesos permite recorrer grafos o árboles (o estructuras dinámicas) diseminando y juntando información. Parecido a heartbeat pero cada proceso hace un único envio y se queda esperando un único receive.

Paradigma 5: algoritmos broadcast Permiten alcanzar una información global en una arquitectura distribuida. Sirven para toma de decisiones descentralizadas. Cosas Distribuidas

Paradigma 6: token passing En muchos casos la arquitectura distribuida recibe una información global a través del viaje de tokens de control o datos. También permite la toma de decisiones distribuidas. Cuando no quiero usar coordinador y quiero solucioón distribuida.

Paradigma 7: servidores replicados Los servidores manejan (mediante múltiples instancias) recursos compartidos tales como dispositivos o archivos.

Manager worker: master distribuye todo de una vez

Token passing
Tengo elementos organizados circularmente