Memoria compartida distribuida

Espacios de tuplas

M. Compartida vs. Paso de mensajes

- M. compartida → comunicación desacoplada:
  - Generador dato (*G*) y consumidor (*C*) totalmente desacoplados:
    - En nombrado: *G* y *C* no se conocen; sólo comparten nombre de dato
    - En el tiempo: no necesitan coincidir
    - Si M. Compartida respaldo en disco → comunicación persistente
- Paso de mensajes → comunicación acoplada:
  - *G* y *C* deben conocerse y coincidir en el tiempo
  - No en el caso de sistemas de colas de mensajes (MOM)
    - Precisamente ese es punto fuerte de MOM
- P. mensajes ofrece sincronización implícita en comunicación:
  - Consumidor se bloquea hasta que se genere dato
- P. mensajes puede manejar heterogeneidad (MPI)
- ¿M. compartida con sincro implícita y manejo heterogeneidad?
  - Espacio de tuplas: **DSM con funcionalidad añadida**

Linda: DSM con espacios de tuplas

- Idea original de Ahuja, Carriero y Gelernter (≈1985)
  - Independiente del lenguaje, SO y HW (M. compartida y SD)
  - JavaSpaces está basado en esta idea
- Modelo de m. compartida en Linda: espacio de tuplas "vivo"
  - Comunicación + sincronización + procesamiento
- Tupla: secuencia de campos con tipo *("hola", 7,3.14, "adios")*
  - existen de forma independiente a sus creadores
  - se acceden asociativamente
- Modelo engloba comunicación y creación de procesos:
  - tupla de datos: de carácter pasivo (base de la DSM)
  - tupla de proceso: de carácter activo (creación de procesos)
    - incluye función en alguno de los campos
    - al insertarla se genera proceso que ejecuta función
    - al terminar: tupla de datos donde resultado función sustituye a función

Linda: Operaciones sobre tuplas

- *out(tupla)*: Añade *tupla* al espacio de tuplas; no bloqueante
- *in(patrón)*: Busca tupla que encaje en *patrón* y la extrae
  - si varias → cualquiera; si ninguna → se bloquea
  - Ejemplo: *in("hola", ?i, ?f, "adios")* → *("hola", 7, 3.14, "adios")*
  - *inp*: versión no bloqueante
- *rd(patrón)*: Igual que *in* pero la tupla no se extrae
  - *rdp*: versión no bloqueante
- *eval(tupla)*: Añade *tupla* de proceso al espacio de tuplas
  - Al insertar se activa proceso que la convertirá en tupla de datos
- Extensión a propuesta original:
  - Múltiples espacio de tuplas
    - Con operaciones para mover/copiar tuplas entre espacios

Linda: Ejemplos de uso

- Semáforo:
  - Bajar semáforo: *in("sem")*
  - Subir semáforo: *out("sem")*
- Cliente/servidor:

```
servidor() {
    int indice=1;
    out("turno", indice);
    while(1) {
        in("peticion", indice, ?pet);
        . . . . . .
        out("respuesta",indice++,res);
    } }
```

```
cliente(){
    int indice;
    . . . . . .
    in("turno", ?indice);
    out("turno", indice+1);
    out("peticion", indice, pet);
    in("respuesta",indice,?res);
}
```

Ejemplos vectores y matrices en Linda

- Iniciar matriz “M” (MxN) con diagonal a 1 y resto a 0

```
for (i=1; i<=M; i++)
    for (j=1; j<=N; j++)
      out(“M”, i, j, (i==j));
```

- Cálculo vector “V” suma de filas de matriz “M” (MxN)

```
for (i=1; i<=M; i++) {
    suma = 0;
    for (j=1; j<=N; j++) {
        rd(“M”, i,  j, ? v);
        suma += v;
    }
    out(“V”, i, suma);
}
```

Usando paralelismo de Linda

```
for (i=1; i<=M; i++)
     out("V", i, suma(i));
.....................
int suma(int f) {
    for (j=1; j<=N; j++) {
       rd("M", f,  j, ? v);
       sum+=v; }
    return sum; }
```

Igual que anterior pero usando una función

```
for (i=1; i<=M; i++)
    eval("V", i, suma(i));
.....................
int suma(int f) {
   for (j=1; j<=N; j++) {
       rd("M", f,  j, ? v);
       sum+=v; }
   return sum; }
```

Versión paralela