Otros mecanismos de concurrencia

Lectura:

Concepts and Notations for Concurrent Programming (Andrews & Schneider)

Manuel Carro

Universidad Politécnica de Madrid

Contenidos

- Múltiples propuestas de creación y sincronización de procesos
- Algunas adoptados en lenguajes de programación
- Otras se han quedado en el tintero (no realmente útiles, difíciles de implementar, etc.)
- Plan:
  - Creando tareas
  - Sincronizando procesos con memoria compartida
  - Sincronizando procesos con memoria distribuida

Creando tareas

- En Ada: tareas como tipos; *elaboración* de variables crea procesos
  - También aparece en otros lenguajes
- `fork/join`:
  - Tareas arrancadas explícitamente dando punto de inicio

```
fork P1;          program P1 is
...               ...
join P1;          end P1;
```

  - Devuelve / espera por identificador tarea
  - Lanzador y lanzado progresan concurrentemente
  - Aparece en librerías Unix: `fork()`/`wait()`
  - También en librerías de *threads*:
    - **POSIX:** `pthread_create()` / `pthread_join()`
    - **Win32:** `CreateThread()` / `WaitForSingleObject()`

Creando tareas (Cont.)

- `cobegin` / `coend`
  - Bloque (como `begin` / `end`) que especifica arranque concurrente:

```
cobegin
  S1; S2; ... Sn;
coend;
```

  - $S_1 \ldots S_n$ se ejecutan concurrentemente
  - `coend` continúa cuando todos los S_i han terminado
  - Variante (replicación tareas):

```
forall I := 1 to n do
  S(I);
end;
```

  - Requisitos fuertes acerca del momento de lanzamiento de tareas

Sincronización con memoria compartida

- Vistos para exclusión mutua:
  - Espera activa (Peterson / Dekker / ...)
  - Semáforos
- Pueden usarse para construir sincronización condicional
- Pero falta de estructuración

Regiones críticas condicionales

- Variables compartidas agrupadas en *recursos*: conjuntos de variables compartidas que necesitan exclusión mutua

```
resource R: V1, V2, ... Vn;
```

- Cada variable a lo sumo en un recurso
- Acceso en exclusión mutua a recursos dentro de regiones:

```
region R when B do S;
```

- R nombre de recurso
- B condición booleana (sin restricción sintáctica)
  - Región suspende si no se cumple
  - Exclusión mutua sobre variables en R
- Caro de implementar (B puede referenciar variables locales)
- Falta de estructuración (acciones sobre recursos dispersas)

Monitores

- Encapsulan estado (privado) y operaciones `Op1, ..., OpN`
- Llamadas a operaciones en exclusión mutua
- Sincronización usando *variables de condición*:
  - **`Wait(cond)`** suspende proceso llamante, abandona exclusión mutua
  - **`Signal(cond)`** permite rearrancar proceso suspendido en *cond*

```
procedure P(A: ...)
  begin
    if not CondP(A, Est)
      then
        Wait(C);
    end if;
    ...
  end P;
```

```
procedure Q(B: ...)
  begin
    ...
    if CondP(Est) then
      Signal(C);
    end if;
  end Q;
```


Sincronización con memoria distribuida

- Basada en alguna forma de paso de mensajes
- No hay variables compartidas $\longrightarrow$ exclusión mutua innecesaria
- Comunicación: mensajes (contienen datos)
- Sincronización: mensajes (espera por llegada / recepción)
- Taxonomía según:
  - Especificación canales
  - Sincronía

Nombrado directo

- Origen y destino especificados en la comunicación
  - Origen especifica destino:

```
send Data to Receiver;
```

- Destino especifica origen:

```
receive Data from Sender;
```

- Se ignoran otros orígenes
- *Pipelines:* una forma (domesticada) de paso de mensajes con nombrado directo `ls | sort`
- Difícil de usar para programar servidores
  - Nombrado global (*mailboxes*)

Llamada a procedimiento remoto (RPC)

- `send` / `receive` usualmente en pares:
  - Pasar datos, recoger resultados
- RPC encapsula ambas direcciones en una sola llamada:

```
call P(args_entrada; args_salida);
```

Especificación de procedimiento remotos

```
remote procedure
  P(args_entrada;
    args_salida)
    begin
      ...
      args_salida := ...;
    end P;
```

- Creado como proceso

Recepción explícita

```
accept P(args_entrada;
         args_salida) -->
  begin
    args_salida := ...;
  end;
```

- Podría ser llamado en cualquier punto
- Sólo en servidor: Rendez-Vous

Panorama