UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II
Scuola Politecnica e delle Scienze di Base

Protocolli

Dispensa

A cura di:

Rocco di Torrepadula Franca

Somma Alessandra

PROTOCOLLI

Quando dobbiamo far comunicare tra di loro entità differenti, che magari devono cooperare verso un obiettivo comune, è necessario utilizzare dei protocolli per regolamentare tale comunicazione. L'obiettivo della progettazione, difatti, non è progettare un singolo componente ma inserirlo in un'architettura, nella quale deve interagire con altri componenti. Nasce allora un problema di sincronizzazione, un problema di protocollo. Possiamo classificare i protocolli in 3 categorie:

- Protocolli sincroni;
- Protocolli asincroni;
- Protocolli semisincroni.

Nei **protocolli sincroni** *le unità lavorano con lo* ***stesso riferimento temporale****, noto e condiviso.* Le due entità non si parlano direttamente: una volta dato il "via", le loro azioni sono tutte motivate dall'esistenza del riferimento temporale comune. Ci sarà solo una fase di "*prologo*" iniziale, per far partire il lavoro. Il problema è che le attività possono sfasarsi e ciò va gestito, talvolta, infatti, conviene utilizzare un protocollo semisincrono.

I **protocolli asincroni** *non sono basati sul concetto di tempo bensì su quello di* ***evento****.* Le entità comunicano alla ricezione di eventi e finchè non si verificano eventi, non trasmettono. Supponiamo, ad esempio, di avere due entità, A e B, che devono comunicare. In particolare, A deve mandare un dato a B. Se utilizzassero un protocollo asincrono, A invierebbe a B al colpo di clock. In questo caso deve avvenire uno scambio di eventi, allora A da il dato (messaggio 1), poi da il via (messaggio 2) e quando B riceve il dato risponde con un messaggio di ok (messaggio 3). A e B hanno eseguito un protocollo, il cosiddetto ***handshaking***: si manda l'informazione, si aspetta di ricevere un messaggio di avvenuta ricezione e in questo modo il sistema converge e termina l'interazione, come mostra la figura 1.

Figura 1: Protocollo handshaking

Un altro protocollo asincrono è l'***handshaking completo***, in questo caso viene aggiunto un quarto messaggio: A manda il dato (messaggio 1), da il via (messaggio 2), B risponde di essere pronto (messaggio 3) e poi invia un ulteriore messaggio per avvertire di aver terminato l'operazione (messaggio 4), come viene mostrato in figura 2.

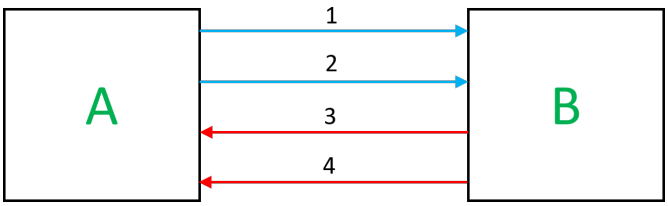

Figura 2: Protocollo handshaking completo

L'intento è fare avere subito ad A una risposta: nel protocollo precedente se B è un'entità meccanica, che impiega parecchio tempo per leggere il dato, A dovrà rimanere in attesa del messaggio 3 più a lungo. Per ricevere il messaggio 3, A deve aspettare che B termini l'operazione; in

questo protocollo, invece, riceve subito il messaggio 3 e al termine delle operazioni riceve anche il messaggio 4. Questo protocollo aggiunge informazioni ma non è detto che quello precedente non funzioni. Infatti, anche se B non comunica di aver finito l'operazione, ovvero senza il filo 4, se A prova a mandare un nuovo dato comunque non riesce, B infatti non è ancora pronto a ricevere il dato successivo. In un protocollo asincrono la base sono gli eventi, in particolare eventi di richiesta, detti ***comandi***, e eventi di risposta, detti ***stati***, perchè rappresentano lo stato in cui si trova la macchina che risponde. Questi protocolli possono essere arricchiti da ulteriori accorgimenti per gestire situazioni patologiche. Supponiamo di star utilizzando l'handshaking, non completo, e che si rompa la linea del messaggio 3. Possiamo inserire un *watchdog* così che, se B non risponde entro un certo tempo, si abortisce il sistema, notificando che la comunicazione non è avvenuta. Ciò non serve a terminare la comunicazione, che si conclude solo alla ricezione del messaggio 3, ma almeno a sbloccare l'entità A che così sa che la comunicazione non può effettuarsi.

Abbiamo visto che **il numero minimo di fili, necessario a un'interazione asincrona, è 3**, possiamo aggiungerne un quarto per avere l'handshaking completo (***protocollo interallacciato***) ma in realtà anche con 3 fili (***protocollo non interallacciato***) funziona ugualmente. Cerchiamo di capire se aggiungere ulteriori fili comporti qualche vantaggio. Supponiamo che l'entità A sia il *master*, M, e l'entità B lo *slave*, S. Il master invia un messaggio, da il via e lo slave risponde di aver correttamente letto il dato. A questo punto allora M sa che l'interazione è avvenuta correttamente, S non lo sa perchè non è certo che M abbia ricevuto il messaggio 3. Potremmo pensare di aggiungere un ulteriore messaggio, un filo, con cui M notifichi a S di aver ricevuto il messaggio 3, in questo modo S sa che l'interazione è avvenuta correttamente ma M no, non è certo che S abbia ricevuto il messaggio. Dovremmo aggiungere un altro filo ma torneremmo al problema opposto. È evidente che **non è possibile arrivare a convergenza**, anche se continuiamo ad aumentare il numero di fili, perchè si tratta di un **sistema distribuito**, *lo stato è distribuito quindi se lo conosce un'entità l'altra non può conoscerlo.* Questo deriva dal fatto che non c'è sincronizzazione, altrimenti non parleremmo di protocollo asicnrono.

Infine, vi sono i **protocolli semisincroni**, che *si comportano come i protocolli sincroni a meno che non si alzi un segnale speciale, un* ***flag***. In un protocollo sincrononо, in cui le entità lavorano con lo stesso riferimento temporale – dunque si parla di *entità isocrone* – potrebbero verificarsi delle situazioni di sfasamento, che vanno gestite. Se A e B comunicano ogni 2 colpi di clock ma per qualche motivo A non riesce ad essere pronta quando dovrebbe, alza un flag per avvertire B e interrompere il protocollo. Per capire meglio il concetto, consideriamo un esempio pratico: il processore e la cache comunicano con un protocollo semisincrono, condividono lo stesso clock e sul fronte di salita il processore richiede un dato, sul fronte di discesa la cache glielo fornisce, in modo sincrono. *Questo è vero fin tanto che la cache contiene i dati richiesti dal processore* ma se si verifica una situazione di *cache miss*, ad esempio a seguito di un salto, la cache impiegherà più tempo a fornire il dato, in quanto dovrà prelevarlo dalla memoria (figura 3).

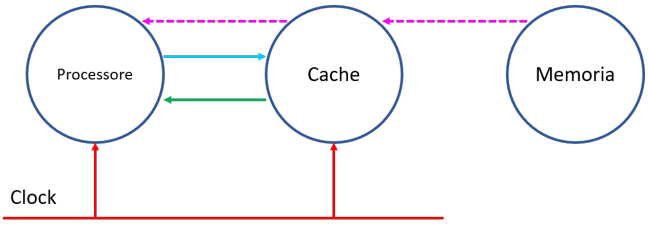

Figura 3: Protocollo semisincrono tra processore e cache

Per segnalare tale situazione, e interrompere la comunicazione sincrona, la cache alza un flag e il processore aspetta; quando la cache preleva il dato, abbassa il flag. **Il processore nota che il flag è stato abbassato, solo al colpo di clock successivo**, per questo è un protocollo semisincrono, non lo sarebbe se lo notasse proprio nel momento in cui il flag viene abbassato.