Training per principianti

TECNICHE DI SINCRONIZZAZIONE

Questa lezione introduce notifier e code come metodi alternativi per passare dati tra processi paralleli

A cura di Alessandro Plantamura

Le variabili sono utili in LabVIEW per passare dati tra processi paralleli. Tuttavia, quando si usano le variabili è spesso difficile sincronizzare i trasferimenti di dati e dovete fare attenzione ad evitare race condition. Notifier e code sono metodi alternativi per passare dati tra processi paralleli. I notifier e le code hanno dei vantaggi rispetto all'uso delle variabili, grazie alla capacità di sincronizzare il trasferimento di dati.

A. VARIABILI

Per mettere in comunicazione cicli paralleli, dovete utilizzare qualche forma di dati condivisi globalmente disponibili. L'uso di una variabile globale rompe il paradigma del flusso di dati in LabVIEW, provoca race condition e causa maggior overhead rispetto al passaggio di dati tramite collegamenti.
Fate riferimento all'Appendice A, Uso delle variabili, per maggiori informazioni sull'uso di variabili per comunicare tra più cicli.
L'esempio mostrato nella figura 1 è un'implementazione meno efficace di un design pattern master/slave. Questo esempio utilizza una variabile, che causa due problemi—non c'è temporizzazione tra master e slave e la variabile può causare race condition. Il master non può segnalare allo slave che il dato è disponibile, così il ciclo slave deve continuamente interrogare la variabile per determinare se il dato cambia.

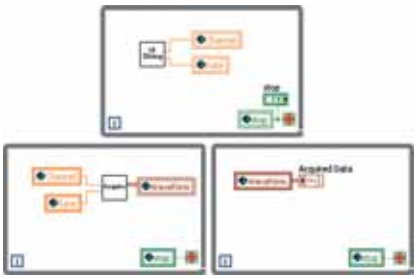

Fig. 1 - Architettura master/slave con uso di variabili globali

B. NOTIFIER

Una più efficace implementazione del design pattern master/slave utilizza i notifier per sincronizzare il trasferimento di dati. Un notifier invia i dati insieme ad una notifica che il dato è disponibile. L'uso di notifier per passare dati tra master e slave elimina qualsiasi problema di race condition. L'uso di notifier procura un vantaggio di sincronizzazione poiché master e slave sono temporizzati quando il dato è disponibile, dando luogo ad un'implementazione elegante del design pattern master/slave. La figura 2 mostra il design pattern master/slave con uso di notifier.

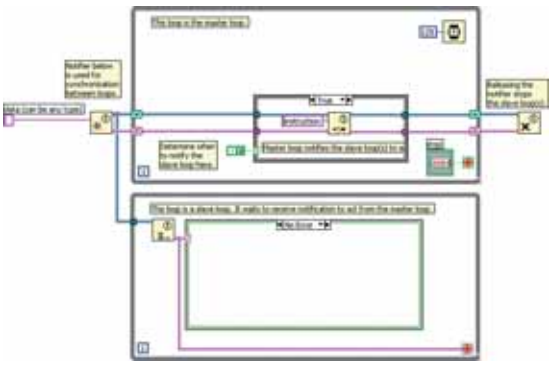

Fig. 2 - Design pattern master/slave con uso di notifier

Il notifier viene creato usando la funzione Obtain Notifier prima che i cicli inizino. Il ciclo master utilizza la funzione Send Notification per inviare notifiche al ciclo slave attraverso la funzione Wait on Notification. Dopo che il VI ha terminato di utilizzare i notifier, la funzione Release Notifier rilascia i notifier.
L'uso di notifier nel design pattern master/slave dà i seguenti benefici:

- Entrambi i cicli sono sincronizzati sul ciclo master. Il ciclo

slave va in esecuzione solo quando il ciclo master invia una notifica.

• Potete utilizzare i notifier per creare dati globalmente disponibili. Così, potete inviare dati con una notifica. Per esempio, nella figura 2, la funzione Send Notification invia la stringa instruction.

• L'uso di notifier crea un codice efficiente. Non avete bisogno di utilizzare interrogazioni per determinare quando il dato è disponibile dal ciclo master.

Tuttavia, l'uso di notifier può avere degli svantaggi. Un notifier non effettua il buffer dei dati. Se il ciclo master invia un altro pacchetto di dati prima che il ciclo(i) slave legga il primo pacchetto di dati, il dato viene sovrascritto e perso.

C. CODE

Le code sono simili ai notifier, tranne che una coda può memorizzare più pacchetti di dati. Di default, le code lavorano in modo first in, first out (FIFO). Perciò, il primo pacchetto di dati inserito nella coda è il primo pacchetto di dati ad essere eliminato dalla coda. Utilizzate una coda quando volete processare tutti i dati inseriti nella coda. Utilizzate un notifier quando volete processare solo il dato corrente.

Quando sono in uso nel design pattern producer/consumer, le code passano i dati e sincronizzano i cicli.

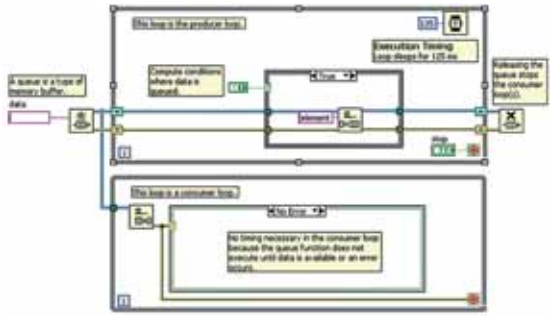

Fig. 3 - Design pattern producer/consumer con uso di code

La coda viene creata usando la funzione Obtain Queue prima che i cicli inizino. Il ciclo producer utilizza la funzione Enqueue Element per aggiungere dati alla coda. Il ciclo consumer elimina i dati dalla coda utilizzando la funzione Dequeue Element. Il ciclo consumer non va in esecuzione fino a quando il dato non è disponibile nella coda. Dopo che il VI ha terminato di utilizzare le code, la funzione Release Queue rilascia le code. Qundo la coda viene rilasciata, la funzione Dequeue Element genera un errore, arrestando di fatto il ciclo consumer. Questo elimina la necessità di utilizzare una variabile per arrestare i cicli.

L'uso di code nel design pattern producer/consumer dà i seguenti benefici:

• Entrambi i cicli sono sincronizzati sul ciclo producer. Il ciclo consumer va in esecuzione solo quando il dato è disponibile nella coda.

• Potete utilizzare le code per creare dati globalmente disponibili messi in coda, eliminando la possibilità di perdere i dati nella coda quando un nuovo dato viene aggiunto alla coda.

• L'uso di code crea un codice efficiente. Non avete bisogno di utilizzare interrogazioni per determinare quando il dato è disponibile da parte del ciclo producer.

Le code sono utili anche per mantenere richieste di stato in una macchina a stati. Nell'implementazione di una macchina a stati che avete imparato, se due stati sono richiesti simultaneamente, potreste perdere una delle richieste di stato. Una coda memorizza la richiesta del secondo stato e la esegue quando la prima è terminata.