

Sistemi per l'elaborazione dell'informazione 2

Flussi e riflussi

Lo scopo di questa esercitazione è la familiarizzazione con i flussi multipli (*multiple threads*) e le primitive di sincronizzazione messe a disposizione da Java.

Processi paralleli e memoria condivisa

Dovete mettere a punto una versione molto semplificata di memoria condivisa da più processi. La memoria è definita da un'interfaccia **Memory** che mette a disposizione due metodi per la lettura e la scrittura: un metodo **read()** che riceve in input l'indirizzo della cella di memoria dalla quale leggere e restituisce l'oggetto letto e un metodo **write()** che riceve in input un indirizzo e un oggetto da memorizzare.

Una prima implementazione di **Memory** consiste nell'utilizzare un array di **Object** di dimensione fissata al momento della creazione. I metodi di lettura e scrittura devono essere sincronizzati (l'accesso alla memoria deve essere permesso a un processo alla volta). Un processo che cerca di scrivere in una cella già occupata attende finché essa non viene liberata. Viceversa, se si cerca di leggere il contenuto di una cella vuota, il processo aspetta finché nella cella non viene scritto qualcosa. Poiché i metodi di lettura e scrittura sono bloccanti, ricordatevi di risvegliare gli altri processi in attesa (usate **notifyAll()** alla fine di tali operazioni). Implementate altresì un metodo sincronizzato **toString()** per visualizzare il contenuto della memoria.

I processi che leggono e scrivono nella memoria saranno istanze della classe **MyProcess** (non possiamo utilizzare **Process** in quanto è già definito in **java.lang**), che permettono di eseguire un flusso (scegliete voi se estendere **Thread** oppure se implementare **Runnable**). Tali processi eseguiranno un algoritmo semplificato, cioè una sequenza di istruzioni di lettura e scrittura in memoria (non ha importanza che cosa viene letto e scritto in memoria; potete assumere che un processo scriva sempre la stessa cosa, per esempio una stringa contenente il nome del processo, che a sua volta è stata specificata come argomento nel costruttore del processo stesso). Al momento della loro costruzione, i processi ricevono in input un nome identificativo, la memoria da utilizzare (sotto forma di un oggetto della classe **Memory**) e un oggetto di tipo **Algorithm**, che contiene la sequenza di istruzione. Il metodo **run()** di un processo dovrà eseguire la sequenza di istruzioni.

Le istruzioni sono oggetti della classe **Instruction**, che contiene un attributo booleano **isRead** (il cui valore sarà **true** se l'istruzione è di lettura e **false** se è di scrittura) e un indirizzo di memoria.

L'interpretazione di un'istruzione è delegata alla classe **Algorithm**, che contiene una sequenza di istruzioni (potete utilizzare un **ArrayList**). La classe **Algorithm** ha un contatore di istruzioni e fornisce un metodo **reset()** che porta il contatore alla prima istruzione da eseguire e un metodo **execute()** che esegue l'istruzione puntata dal contatore e incrementa quest'ultimo. Tale metodo riceve in input la memoria su cui deve essere eseguita l'istruzione corrente e l'eventuale oggetto da scrivere nel caso di un'istruzione di scrittura, e restituisce un intero: **-1** se l'istruzione eseguita è l'ultima della sequenza e **0** altrimenti.

Esempio: L'istruzione **WRITE 10** è definita come un oggetto della classe **Instruction** contenente **false** nell'attributo **isRead** e **10** nel campo relativo all'indirizzo. Se il contatore di **Algorithm** punta a questa istruzione, l'invocazione di **execute(mem, obj)** scrive **obj** nella memoria **mem** all'indirizzo **10**. L'istruzione

`READ 10`, invece, ha l'attributo `isRead` impostato a `true`. In questo caso, la chiamata di `execute(mem, obj)` ignora `obj` e stampa a video l'oggetto contenuto in `mem` nella cella all'indirizzo 10.

Deadlock

Provate a inizializzare i processi con algoritmi che generano o meno deadlock. Di seguito ci sono alcuni esempi:

- Due processi vogliono scambiarsi un'informazione: il processo A legge il contenuto della prima cella e lo scrive nella seconda, mentre il processo B legge dalla seconda e scrive nella prima. Che cosa succede se entrambi vogliono prima ricevere l'informazione e poi scrivere in memoria?
- Il processo A scrive n oggetti in una memoria di capienza n , dalla cella 0 alla cella $n - 1$. Il processo B legge oggetti dalla cella $n - 1$ alla cella 0. Che cosa succede in memoria?
- Provate a generare due sequenze di istruzioni che scrivono e immediatamente leggono in una cella scelta a caso. Notate come cambia la probabilità di avere deadlock in funzione della lunghezza delle sequenze e della capienza della memoria.

Suggerimento: Per vedere i diversi flussi all'opera, fate sì che la chiamata di `execute()` nel metodo `run()` del flusso sia ritardata di un tempo variabile tra 0 e 100 millisecondi.

Esercizio facoltativo

L'implementazione di `Memory` precedentemente proposta rende sincronizzati i metodi `read()` e `write()`. In questo caso, ogni volta che un processo legge o scrive in memoria, l'intero array viene bloccato. Questa situazione è ovviamente poco efficiente, poiché vorremmo che fosse possibile leggere e scrivere contemporaneamente in celle diverse della memoria.

Un'implementazione più efficiente di `Memory` (chiamiamola `FineGrainedMemory`) può essere effettuata tramite un array di oggetti della classe `Cell`. Ogni oggetto di questa classe contiene un attributo di tipo `Object` (l'effettivo contenuto della cella) e fornisce due metodi *sincronizzati* `read()` e `write()` che vengono invocati dai metodi (questa volta *non sincronizzati*) `read()` e `write()` di `FineGrainedMemory`.

Un altro esercizio facoltativo

Scrivete un'applicazione che permette di visualizzare tramite un'interfaccia grafica la memoria condivisa, in forma di un insieme di rettangoli (uno per ogni cella), disposti orizzontalmente o verticalmente (scegliete voi). L'interfaccia grafica deve mostrare quel che succede ai contenuti della memoria durante l'esecuzione di un algoritmo, scrivendo dentro alle celle il numero in esse contenuto e mostrando in modo sensato le celle vuote (per esempio colorandone lo sfondo).