Esercitazione #1 -- Corso di Sistemi Operativi

Multithreading in Java

Luca Gherardi e Patrizia Scandurra – a.a. 2012-13

1. **Esercizio (*verificare il grado di concorrenza della piattaforma ospitante*)**: Creare un'applicazione con due thread figli. Ciascuno dei due thread ha associato un simbolo (ad es. * o #). Sovrascrivere il metodo run() in modo che ciascun thread stampa continuamente a video il proprio simbolo, andando a capo ogni 50 simboli stampati.

2. **Esercizio (Somma)** Scrivere un programma multithread in Java che esegue una funzione di somma tra due numeri interi inseriti dall'utente. Creare un thread separato per effettuare la somma e restituire il risultato in una variabile intera (globale) *condivisa* con il thread principale. Il thread principale deve sincronizzarsi sulla terminazione (*join*) del thread figlio *Somma* e stampare poi il risultato a video.
   [Suggerimento: Per la variabile condivisa, creare una classe involucro HoldInteger con dei metodi get/set.]

3. **Esercizio**: creare un'applicazione Java contenente 3 thread come segue. Un thread principale corrispondente al metodo main() che stampa a video il suo nome, e poi crea ed avvia due thread dello stesso *tipo*: runner1 di nome "Pippo" e runner2 di nome "Caio". Il comportamento dei thread consiste nel stampare a video il proprio nome. Provare ad eseguire l'applicazione, e a comprendere cosa succede.

4. **Esercizio**: creare un'applicazione Java contenente 3 thread. Un thread principale (main()), che stampa a video il suo nome, crea ed avvia un thread runner1 di nome "Pippo" un thread runner2 di nome "Caio" dello stesso tipo di runner1, creato a sua volta da runner1 nel metodo run() dopo aver stampato a video il proprio nome. Provare ad eseguire l'applicazione. Cosa succede? Provate ad introdurre un ritardo (con Thread.sleep) di 100 ms dopo ogni stampa a video, e rieseguire l'applicazione. Cosa succede?

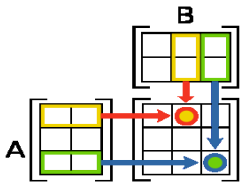
5. **Esercizio**: creare un'applicazione Java contenente 2 thread: un thread principale (main()) crea e avvia un nuovo thread, e poi ogni 500 ms controlla se il figlio è ancora vivo. Se il figlio ha terminato, termina anche lui. Il processo padre deve produrre varie stampe a video per descrivere l'attività di testing, ed in particolare deve stampare il numero di controlli che effettua. Il thread figlio effettua per *n* volte una generica attività *A* di una certa durata. Provare ad eseguire l'applicazione. Cosa succede? Provate a variare la durata di *A* ed il parametro *n*, ed eseguite di nuovo.

6. **Esercizio (MatrixSum).** Si vogliono sommare tutti gli elementi di una matrice intera NxM con un insieme limitato N di thread (un thread per riga) che sommino i numeri delle righe in parallelo. Il thread principale deve sincronizzarsi sulla terminazione dei thread figli (*join multiplo*) e stampare poi il risultato finale (la somma dei totali delle righe) a video.

7. **Esercizio (Fibonacci)** Scrivere un programma multithread in Java che generi la successione di Fibonacci in base ad una certa quantità N di numeri specificati dall'utente. La successione di Fibonacci è una successione di numeri interi naturali definibile assegnando i valori dei due primi termini, $F_0 := 0$ ed $F_1 := 1$, e chiedendo che per ogni successivo n>1 sia $\mathbf{F_n = F_{n-1} + F_{n-2}}$
   Ad es. per N=10, la successione di numeri di Fibonacci è: 0,1,1,2,3,5,8,13,21,34.
   Il programma deve creare un thread separato che scrive i numeri di Fibonacci.

8. **Esercizio (Prodotto di Matrici)**. Scrivere un programma multithread in Java che calcola il prodotto C (m×p) tra due matrici A (m×n) e B(n×p) così definito:

$$(A \times B)_{ij} = \sum_{r=1}^{n} a_{ir} b_{rj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}$$



per ogni i=1..m e j=1..p.

Il programma deve calcolare ogni elemento C(i,j) in un thread separato (ciò comporta la creazione di m×p thread). Testare il programma con il seguente esempio:

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 1 & -3 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 2 & 5 & 1 \\ 0 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 4 \\ 2 & 11 & -2 \end{bmatrix}$$

A (2x3) B (3x3)

*[**Suggerimento**: Usare due cicli (doppi con i,j) separati: uno per lo start di mxp thread, ed uno per il join su mxp thread t(i,j).]*