Commento alle soluzioni proposte

Esercizio di programmazione concorrente

Per realizzare il comportamento richiesto occorre che i vari thread (produttori e consumatori) comunichino al main quando stanno per terminare.
Il meccanismo degli interrupt non è adatto, perché il main ricevendo un iterrupt non saprebbe se arriva da un produttore o da un consumatore.
Una soluzione semplice ed efficace consiste nell'implementare una classe (`Signal` nel codice proposto) attraverso la quale i thread produttori e consumatori comunicano la propria terminazione. Il main può quindi sospendersi in attesa che qualcuno termini chiamando un opportuno metodo di questa classe (`waitSignal` nel codice proposto). `waitSignal` restituisce anche il tipo (produttore o consumatore) del thread terminato.
Nel codice proposto, i consumatori chiamano il metodo `incConsumers` per svegliare il main, mentre i produttori chiamano `incProducers`.

Esercizio di programmazione distribuita con socket

In base alla consegna data, occurre definire un server che

1. accetta una connessione da un client
2. iterativamente:
   a) manda al client un segnale
   b) aspetta un po'

Il client

1. si connette al server
2. iterativamente:
   a) aspetta un input dal server
   b) quando lo riceve, reagisce creando un thread child

Esercizio di programmazione distribuita con RMI

Il comportamento desiderato è lo stesso del sistema realizzato con socket.
In questo caso, l'unico problema da risolvere è il fatto che il server deve avere un riferimento al client per potergli mandare il segnale cui il client reagirà attivando un nuovo child. Con RMI, il problema si risolve mediante il meccanismo del callback: il client deve essere a sua volta un oggetto remoto, in modo da poter comunicare al server un suo riferimento. In tal modo, il server ha un riferimento remoto al client, che può usare per inviargli i segnali.
Dunque, il server (che deve essere un oggetto remoto) deve:

1. Mettere a disposizione un servizio (`subscribe` nel codice dato) attraverso il quale i client comunicano il proproprio riferimento remoto
2. Registrare il proprio riferimento sul registry
3. Iterativamente, effettuare una chiamata al metodo opportuno (`vai` nel codice dato) dei client.

Il client (che deve essere un oggetto remoto) deve

1. Mettere a disposizione un metodo (`vai` nel codice dato) che causa la creazione di un nuovo thread child
2. Recuperare dal registry il riferimento remoto al server
3. Comunicare al server (mediante metodo `subscribe`) il proprio riferimento remoto